Naval Research Laboratory

Washington, DC 20375-5320

NRL/FR/5510--01-9964

A Language Use Approach to Human-Computer Interaction

DEREK P. BROCK

Navy Center for Applied Research in Artificial Intelligence

January 31, 2001

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave Blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 31, 2001	

4. TITLE AND SUBTITLE

A Language Use Approach to Human-Computer Interaction

5. FUNDING NUMBERS

PE-0601153N
TA-R4207

6. AUTHOR(S)

Derek P. Brock

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Research Laboratory
Washington, DC 20375-5320

8. PERFORMING ORGANIZATION REPORT NUMBER

NRL/FR/5510--01-9964

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research
800 North Quincy Street
Arlington, VA 22217-5660

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

In this report, elements of Herbert Clark's theory of language use as a form of joint action are examined as a paradigmatic basis for the design of human-computer interaction. Particular attention is given to the theory's characterization of the function and nature of common ground and the form of joint actions in communicative acts. This is balanced with a presentation of work to develop an application user interface that employs a cognitive simulation to represent portions of user-computer common ground computationally as it accumulates in the joint activity of its underlying application task. It is argued that some form of computational modeling of common ground is imperative in the design of simulated cognition for interactive purposes.

14. SUBJECT TERMS

ACT-R	Cognitive modeling common ground	Human-computer interaction
Intelligent user interface	Joint actions	Joint projects
Language use	Levels of action	Simulated cognition
Task analysis	Task modeling tracing	

15. NUMBER OF PAGES

46

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std 239-18
298-102

CONTENTS

A LANGUAGE USE APPROACH TO HUMAN-COMPUTER INTERACTION

1. INTRODUCTION

1.1 User-Computer Communication

Users regularly face the challenge of getting a computer to do what they want it to do. This can mean anything from having to locate an obscure program feature, to wrestling with an elaborate series of interactions, to simply not knowing what to do next. Although it is true that such problems often arise in part from a lack of attention to various human factors and usability principles in a program's design, the fact remains that even the best user interfaces have fundamental shortcomings that are directly related to the limits of their abilities to communicate well with users in the sense that people are generally able to communicate well with each other (cf. Norman, 1992, ch. 11). For instance, programs are rarely designed to actively anticipate their users' varying concerns, or to monitor and recognize what their users are doing or are trying to do. Nor are they routinely able to summarize an activity, much less to keep track of it in a meaningful way in order to verify a user's understanding of one part or another. Even with the benefit of carefully designed aids, such as user manuals and online help (e.g., Shneiderman, 1998 and Sellen and Nicol, 1990), users in situ very often find they are on their own when it comes to puzzling out a program's more elaborate features and abilities, or when they must grapple with a new or difficult task. In complex or mission-critical systems, a program's communicative shortcomings may very well be one of its most crucial deficiencies.

Nevertheless users and computers do form a true communicative partnership because interactions between them rely on an exchange of information. Human-computer interaction amounts to a language in which, at a minimum, computers are told what they are to do and users expect to be told what the computer has done* (in fact, much more than just this occurs). Consider that the characteristic activity people and computers engage in is *information processing* (Simon and Kaplan, 1989). Indeed, computers are remarkable for the breadth of interactivity they can support and the fact that their processing is fundamentally computational, as opposed to mechanistic. More importantly though, computers, like people, are able to maintain flexible representations of knowledge, and this sets them apart from all other complex devices of human design (cf. Kay, 1984).

How then are the communicative shortcomings of user interfaces to be improved? As the power of computer systems has grown, there has been a surge of interest in approaches to this problem. It has, for instance, been widely pointed out that communication between people is regularly supported by actions occurring in nonlinguistic channels, and that there are many potential advantages to exploiting such modes of expression in human-computer interactions (e.g., Sullivan and Tyler, 1991; Laurel, 1990; and Maybury, 1993). What is most important though is not the medium of interaction—natural language, graphical display, etc.—but the nature of the information processing that supports the content of interactions. It is hardly necessary for people and computers to literally speak to each other, but it would be a significant step forward for user interfaces to be able to remember, reason about, and respond to interactions much as a person would.

Hence this report proposes that a large part of the answer lies directly in our understanding of how people communicate with each other and what it is that they do when they use language. In his 1996 book, *Using language*, Herbert Clark (1996b) makes a compelling argument for the claim that "language use is really a form of *joint action*." In joint

*The working definition of the human-computer interaction research group at the U.S. Naval Research Laboratory reads, "[Human-computer interaction] can be viewed as the bi-directional communication of information between two powerful information processors: people and computers." (http://elazar.itd.nrl.navy.mil)

Manuscript approved May 17, 2000

actions, people coordinate with each other to accomplish shared purposes that are part of their broader ends in *joint activities*. In particular, all joint activities advance through the accumulation of *common ground*—the knowledge, beliefs, and suppositions that people bring to, keep track of, and share about an activity. Clark's thesis sets out a coherent theoretical framework for understanding much of what communicative acts involve, and in the work presented here, human-computer interaction is approached not as a device use problem (Norman, 1986) but as a type of bona fide joint activity in which humans and computers are viewed as the participants.

This report's title, "A Language Use Approach to Human-Computer Interaction," is chosen to convey, in the broadest sense, what enters into any form of joint activity. Clark describes joint activities as a basic category that encompasses all participatory circumstances in which conventional language plays a role. More to the point, he notes, "If we take language use to include such communicative acts as eye gaze, iconic gestures, pointing, smiles, and head nods—and we must—then all joint activities rely on language use" (Clark, 1996b, p. 58). For Clark, language in its linguistic sense is simply one of many possible signaling systems, some highly organized and others spontaneously improvised. The insight is that coordination in all purposeful joint activities requires some form of signaling. And indeed, much of what constitutes the public character of human-computer interaction is merely a set of signals derived from acts of pointing and clicking, typed commands entered through a keyboard, and processed information displayed on a screen.

But what does it really mean to take a language use approach to human-computer interaction? What distinguishes joint activities from other endeavors is common ground. Evidence of it is present in everything people design to be used by others and in everything people do together. What is being proposed by saying that human-computer interaction can be viewed as a type of joint activity is that the computer as an information processing agency is to be taken seriously in its role as a communicant. Well designed application user interfaces *exhibit* their common ground with users through their use of metaphor, consistency, affordances, and so on (e.g., Erickson, 1990; Tognazzini, 1990; and Norman, 1988). In addition, they should be designed to *support* their common ground with users. People accumulate and maintain common ground so that they have a means—a language—for coordinating and advancing their activities with others. Ideally, it should be no different in the design of human-computer interaction. To the extent that it is possible or practical, the basis of joint activities between humans and computers should be designed to resemble our understanding of the basis of joint activities between people.

Common ground is both knowledge and process. As a knowledge basis for the design of conventional user interfaces, a determination of user-computer common ground should influence all facets of the display and the interaction design. The designer, through an analysis of the task domain as a joint activity, should think in terms of what every component of the user interface will communicate to the user, and seek to anticipate information the user will need to coordinate the task activity with the computer (cf. Norman, 1992, ch. 17). User-centered design approaches (Norman, 1986) strive for a similar result, but not through a principled interrogatory framework based on the notion of common ground. Menus, dialogues, entry points, help messages, and so on, all serve as coordination devices—as a language—offered by the user interface for carrying out the joint activity of the task. When considered in terms of common ground, the shortcomings and strengths of the design of these components are more clearly revealed and understood.

Common ground also has clear process implications for user interfaces. As joint activities progress, common ground accumulates—at least in users it does. Remarkably, it also accumulates in ordinary programs, to some degree, but seldom in ways that resemble its characteristics in people. In people, common ground develops entirely in cognition, whereas in conventional user interfaces, it accumulates through what are at best ad hoc and often obscure processes such as preferences, changes in the display, and/or history and undo mechanisms. Ideally, users should be able to confer with a program at any time and get responses that are perceptive and relevant. What user interfaces need are additional mechanisms that serve this function, whose role is to interpret and facilitate what is going on in the joint activity in ways that have meaning for the user.

Interfaces incorporating such mechanisms are, in fact, already beginning to appear, not only in research laboratories (e.g., Rich and Sidner, 1998), but also in commercial products (e.g., Horvitz et al., 1998). The role of common ground is so intuitive and fundamental in interactive activities that it is always present in one form or another, whether or not it has been fully recognized or accounted for in a design. In this view, so called "intelligent user interfaces" are interfaces that take the computer's potential role in task activities seriously enough to have been crafted to go

significantly beyond conventional reflexive or stimulus-response designs. By employing computational mechanisms that make context sensitive inferences or that recognize patterns of behavior, and so on, these interfaces accumulate and use aspects of common ground in ways that users presumably recognize as being more consistent with their own internal representations of activities.

Nevertheless, it is the process component of common ground in user interfaces that is least understood and often least appreciated, and at present, there are only a few candidate mechanisms for its implementation. In this report, the problem is examined through the computational paradigm of cognitive modeling, the presumption being that a theory of human cognition is relevant to a theory of language use between people. Although there are practical strengths and weaknesses to this particular computational strategy, it does provide clear evidence that a language use approach to human-computer interaction is merited.

1.2 Guide to the Report

The work presented here draws in multidisciplinary fashion from three different fields that are themselves multi-disciplinary—human-computer interaction, processes in language use, and cognitive science. It hardly needs saying that the challenges faced in each of these areas of inquiry are substantial, and that in each there is still much that is as much art as science. As a student and a researcher, my introduction to cognitive modeling and various results in cognitive science led to my interest in cognitively augmented user interfaces. My subsequent introduction to Clark's body of work provided a coherent framework—a language—for crystallizing these ideas and characterizing the issues.

This report has two substantive goals. The first, as background, is to provide an introductory review of Clark's theory of language use as joint action, choosing for emphasis material that is relevant to the nature of human-computer interaction. Some of the implications of Clark's work in the context of user interfaces are briefly examined as part of this review. All of this material is presented in Section 2. The second goal, presented in Section 3, is to describe a cognitive modeling approach to the active representation of common ground called "task model tracing." In task model tracing, a computational theory of human cognition known as "ACT-R" (Anderson and Lebiere, 1998) is used by an application user interface as the basis for simulating the accumulation and use of elements of user-computer common ground in the joint activity of a portion of the application's task. Section 4 of the report presents a high-level summary review and a description of future work.

2. CLARK'S CHARACTERIZATION OF LANGUAGE USE

2.1 Introduction

In his monograph *Using language* (1996b), Herbert Clark gives a persuasive account of language use as a form of *joint action*—what emerges when people coordinate their individual actions with others. Coordination requires communicative acts to come into play, and this leads to notions of *common ground*, *meaning and understanding*, *levels of action*, and to the essence of what language itself is, *signaling*. People do things together, and yet they act as individuals. Ultimately in the study of language use, the notion of joint actions cannot be separated from the individual actions that comprise them.

This section presents a summary introduction to Clark's notions about the nature of language use as background material that is intended to motivate the modeling work presented in Section 3. With the exception of Section 2.7, all of the material in this part of the report (i.e., Section 2) has been drawn from Clark (1996b). Aspects of the theory that are deemed relevant to the nature of human-computer interaction have been emphasised, although this has not been generally noted in context to preserve the clarity of the exposition. The presentation is greater than should be necessary to apprehend the immediate goals of the task model tracing effort, but its length is merited by the richness of the theory, and the belief that the additional material will be useful in subsequent modeling and/or design efforts.

2.2 Overview of Language Use

Clark begins by remarking that "Language is used for doing things." And it is. It is used to carry out what people do together, when they are with each other and when they are apart. It is used to do anything that has a social purpose.

In doing things together, people participate in a special kind of activity, a joint action. Joint actions occur whenever individuals choose to act in some form of coordination with each other, regardless of the details or the circumstance. Just as two people waltzing or shaking hands constitute joint actions, so do the activities that emerge when people correspond or simply converse.

Activities involving language use occur in a wide range of spoken and written settings, or some mix of the two. Settings can be characterized in various ways, as personal, fictional, mediated, and so on, but the most basic setting of language use is always *face-to-face conversation*. Nonbasic settings lack one or more of the natural features of face-to-face conversation related to immediacy, medium, or control that are listed in Table 1. When any of these features are missing, people are always forced to resort to using different skills and procedures to accommodate their absence.

Table 1 — Features of Face-to-Face Conversations

Features related to *immediacy*:		
1	Copresence	The participants share the same physical environment.
2	Visibility	The participants can see each other.
3	Audibility	The participants can hear each other.
4	Instantaneity	The participants perceive each other's actions at no perceptible delay.
Features related to *medium*:		
5	Evanescence	The medium is evanescent—it fades quickly.
6	Recordlessness	The participants' actions leave no record or artifact.
7	Simultaneity	The participants can produce and receive at once and simultaneously.
Features related to *control*:		
8	Extemporaneity	The participants formulate and execute their actions extemporaneously, in real time.
9	Self-determination	The participants determine for themselves what actions to take when.
10	Self-expression	The participants take actions as themselves.

In any setting where language is used, people variously assume the roles of *speaker* and *addressee*.* Others may be present, but not as addressees or participants. Speakers intend for their addressees to understand an address and to act on that understanding. They know further that this depends entirely on addressees taking a number of actions on their own. Addressees know they have to listen, watch, and otherwise attend when they are being addressed, and that they must try to make sense of this information and understand it. In managing to coordinate all of these actions with respect to each other, what emerges is both a joint action *and* a use of language.

Joint actions are, by definition, social actions, and social actions cannot be coordinated without a basis of shared knowledge—common ground. When language comes into play, not only is the action itself coordinated, but so also are the notions of *speaker's meaning* and *addressee's understanding*. Speakers convey meaning with *signals*—any conspicuous action deliberately performed for others—their addressees—to identify. Speakers devise their signals out of what they believe is common ground, intending that their references to it will be recognized by their addressees; in this way, meaning and understanding arise.

Language use also has *layers of action*. All of the joint actions that speakers and addressees carry out as themselves take place in the primary layer, whereas actions with meaning that transcends the participants' literal circumstances occur in a secondary layer or higher. This notion of layering accounts for the creation of conversational roles for persons not present and similarly for removes in time and/or place.

* A terminological note: Much of what is ordinarily called "language" is inescapably linguistic in character, and Clark frequently uses linguistic examples and the term *speaker* to illustrate a point. Nevertheless, he notes that, "At least in the notion of 'language use,' [language] must include every method by which one person means something for another." Hence, by *speaker*, he intends, more broadly, the notion of *signaler* or *presenter*.

A distinguishing characteristic of the actions individuals take in joint actions is their *participatory* as opposed to *autonomous* nature. Participatory actions inherently make reference to the actions of other participants. The individual actions themselves may be, and often are, very different from each other, but they nevertheless lack full autonomy.

The joint actions that result when speakers and addressees use language as themselves in the primary layer of action are not just atomic events. Instead, they are what emerge from a process that is made up of *levels* of tightly paired, subsidiary joint actions. Sounds must be vocalized and heard, utterances must be formulated and identified, and meaning must be intended and understood. Component joint actions build upon each other, and each has a specific role that is essential in the overall process of language use. In nonbasic settings, these levels of joint action can and do become decoupled. The joint action in which speaker's meaning and addressee's understanding arise, though, is privileged—it is what defines language use.

Coming full circle, a further characteristic of language use is a product of the fact that language is used for doing things. Deliberate actions have consequences that are both intended and also unintended. When consequences are intended, the product is said to be *anticipated*; when they are not, the product is said to be *emergent*. Many of the regularities of language use emerge unanticipated and unintended. In Table 2, six propositions summarize Clark's working assumptions in this overview of language use.

Table 2 — Six Propositions about Language Use

Proposition 1.	Language fundamentally is used for social purposes.
Proposition 2.	Language use is a species of joint action.
Proposition 3.	Language use always involves speaker's meaning and addressee's understanding.
Proposition 4.	The basic setting for language use is face-to-face conversation.
Proposition 5.	Language use often has more than one layer of activity.
Proposition 6.	The study of language use is both a cognitive and a social science.

2.3 Doing Things Together—Joint Activities

Language *use* in joint activities is secondary to the activity itself; it is a means to an end, an emergent product, not what the activity is *about*. And yet the two are inseparable. People cannot do things together without communicating—using language in the broadest sense—and this means that one cannot be understood without the other.

Joint activities are fundamentally different from what people do on their own autonomously. They are social activities, bounded in time or ongoing, but distinguished by having more than one participant. The many ways in which joint activities differ can be thought of as varying on a variety of dimensions such as scriptedness, formality, and verbalness. Two dimensions that are particularly useful in understanding what goes on in language use are cooperativeness, the degree to which an activity is collective vs adversarial, and governance, the degree to which an activity is egalitarian vs autocratic.

2.3.1 Some Generalizations

In doing things together, people assume *activity roles*. A nonparticipant in one activity may be a participant in an enclosing activity. Roles arise from an activity's nature and ratify participation. Along with personal identities, they shape much of the making of an activity—who does what and why.

Joint activities serve the goals their particpants pursue and hope to achieve. Some goals are public and others private. Each has consequences for the conduct of the activity. More often than not, a variety of goals are being pursued at the same time, some related and some not. Joint activities usually have an obviously dominant goal, but their participants may also have procedural and interpersonal goals, among others. Like any other information, goals become *public* when they become openly recognized in one form or another. A joint activity is defined by its joint goals, and these are always public. *Private* goals are kept from others largely for reasons of expedience.

How people coordinate what they do with each other is central to how language is used. In joint activities, people do this with mixes of conventional and nonconventional procedures that are themselves made up of fixed and/or negotiated hierarchies of smaller joint activities or actions. For each part or phase of the process to work, participants need to be mutually confident of each other's engagement, and this means contriving their entries and exits from one part to the next and ultimately for the whole activity. Any of a variety of dynamics may also come into play. Activities overlap, pause, divide, expand, and so on. Table 3 summarizes the points made in this section.

Table 3 — Some General Claims about Joint Activities

Participants	A joint activity is carried out by two or more participants.
Activity roles	The participants in a joint activity assume public roles that help determine their division of labor.
Public goals	The participants in a joint activity try to establish and achieve public goals.
Private goals	The participants in a joint activity may try to achieve private goals.
Hierarchies	A joint activity ordinarily emerges as a hierarchy of joint actions or joint activities.
Procedures	The participants in a joint activity may exploit both conventional and nonconventional procedures.
Boundaries	A successful joint activity has an entry and exit jointly engineered by the participants.
Dynamics	Joint activities may be simultaneous or intermittent, and may expand, contract, or divide in their personnel.

Reprinted (with modifications) from Clark, *Using language*, pp. 37–38, ©1996 Cambridge University Press

2.3.2 A First Look at Common Ground

Clark contends that all joint activities advance through their participants' accumulation of common ground—the knowledge, beliefs, and suppositions each assumes they share about an activity. The study of common ground and its accumulation has previously been limited to its occurrence in discourse. The broader view—that it accumulates in all joint activities—follows naturally from the notion that language use is a form of joint action.

People bring presuppositions to their participation in joint activities. Their public actions during an activity and possibly other occurrences in the activity's environment can be thought of as state-changing events. Not all states and events, though, are taken by the participants to be part of the activity in an official sense. As the state of an activity changes, its participants' presuppositions are cumulatively modified. An activity's common ground is made up of these presuppositions, and at any point in most activities they can be divided into the three parts described in Table 4.

Table 4 — Three Parts of Common Ground

1. *Initial common ground.* This is the set of background facts, assumptions, and beliefs the participants presupposed when they entered the joint activity.
2. *Current state of the joint activity.* This is what the participants presuppose to be the state of the activity at the moment.
3. *Public events so far.* These are the events the particpants presuppose have occurred in public leading up to the current state.

Reprinted (with modifications) from Clark, *Using language*, p. 43, ©1996 Cambridge University Press

The first part of common ground—initial common ground—can be, and usually is, vast. One of its most useful characteristics in many situations is its participants' shared knowledge of applicable information, such as cultural norms and procedures.

People's ongoing sense of the second part of common ground—the current state of the joint activity—begins from the moment they enter into the activity itself. This sense of state includes the participants' awareness of who is doing

what and the status of their various goals in the joint activity. This part of common ground is frequently aided from moment to moment by the actual state of objects in the participants' immediate physical environment. Beyond their simple physical existence, objects and scenes in the world are routinely used as *external representations* of what is often a highly developed, shared understanding. Board games capture the essence of the idea. Table 5 describes some of the properties of external representations that make them so useful in joint activities. Three other points can be made about external representations. Because they are features of the physical environment, they are highly reliable for purposes of joint reference. They are also useful for participants as a mental aid, not only for purposes of recall, but also as a medium for reasoning forward. Finally, because of their tangible nature, external representations are often the medium of the joint activity itself.

Table 5 — Some Properties of External Representations

1. *Physical model.* The scene of a joint activity can be used by its participants as a physical model of the activity that can be viewed, touched, and manipulated.
2. *Markers.* Features of and in external representations can be markers that denote elements of the joint activity.
3. *Location interpretation.* The spatial location of markers in an external representation can be a factor in the interpretation of these markers.
4. *Manipulability.* The movement or alteration of markers can also be a factor in their interpretation.
5. *Simultaneous and parallel accessibility.* All participants generally have access to and can consult the scene of a joint activity.

Paraphrased from Clark, *Using language*, pp. 46–47,

In keeping track of the third part of a joint activity's common ground—the public events so far—people rely on what they already know about what they are doing to help them identify what has happened. The result is an *annotated record* of single events and purposeful sequences expressed in personal terms. As events recede, people go on to form more broadly expressed *outline records* by abstracting away details.

Pragmatically, people keep track of common ground in joint activities so that they have a reasonable sense of what their counterparts know. But people's internal representations of things are common ground only to the extent that they correspond with each other. What doesn't correspond isn't common ground. Inevitably, representations of common ground fail to correspond in many ways. Often these discrepancies go undetected, but when they are discovered, people tend to correct them immediately to avoid coping with the consequences.

2.3.3 Language in Joint Activities

Conventional language, whether spoken or written, is used regularly in the conduct of joint activities. But verbalness—the degree of an activity's linguistic character—is simply a dimension by which joint activities can vary. A telephone conversation is almost entirely a linguistic activity whereas waltzing is almost entirely nonlinguistic. These two extremes suggest a kind of *discourse continuum*. Somewhere in the middle it becomes undeniably apparent that a text of the verbal component in joint activities cannot meaningfully stand alone. Without a record of the larger joint activity, any words participants use begin to lose a portion of their coherence. When conventional language is separated from the circumstances in which it occurs, the result is artificial—something is lost. The full record of a joint activity is a full record of all of its communicative acts—all of the signaling and coordination that occurs, as well as any conventional language that comes into play. If conventional language is taken to be simply one of many possible forms of communicative acts, acts in which speaker's meaning and addressee's understanding play a role, a broader understanding of the notion of language use emerges. Language use can now be seen as present throughout the discourse continuum and, indeed, all joint activities can be seen to depend on it.

2.4 Coordinating Content and Processes—Joint Actions

When people decide to do something together, it becomes necessary for them to coordinate their actions as individuals with respect to each other. What emerges from these acts of coordination are sequences of *joint actions*—the

components of joint activities. To carry out a joint action, the action's participants must work out both what they intend to do and how they are going to do it. More often than not, each of these considerations affects the other in an opportunistic way. The cognitive and/or physical *processes* each person employs usually depend on the joint action's *content*—what it is they are trying to do together. Similarly, the content itself may have to be reshaped, depending on what processes are available. In these respects, the nature of joint actions is the nature of language use.

2.4.1 Participation and Coordination

In Clark's view, what technically distinguishes joint actions from other kinds of actions is their *participatory* nature. Joint actions are participatory in the sense that more than one person is involved and each is pursuing the same purpose in a cooperatively coordinated manner. According to this view, a joint action's participants must all intend to be doing their respective parts and believe that the joint action itself includes both their own and each other's intentions and beliefs. Noncooperative or deceptive actions between people are not taken to be joint actions (even though they may involve the participation of more than one person, such as in a game), and neither are actions in which an individual acts autonomously with respect to the actions of another.

2.4.1.1 Coordination Problems

What is key in joint actions is the element of coordination. Joint actions are really solutions to *coordination problems* that come about when people's concerns are in some way coincident and meeting them requires cooperative action. Indeed, this is the essence of language use. To solve coordination problems, people require a jointly salient *basis* for coordinating their expectations of each other. This basis, referred to as a *coordination device*, can be virtually anything that will allow the participants in a joint action to arrive at a mutual supposition or expectation of what their own and each other's part in the action should be. Such mutual expectations arise from the participants' current common ground. A robust coordination device is something that is not only known to all of the people facing a coordination problem, but also indicates to all of them that they all know that something, and that that something indicates their mutual expectation. The role of salience and common ground in solving coordination problems leads to a *principle of joint salience*—The solution to a coordination problem among two or more agents is the solution that is most salient, prominent, or conspicuous with respect to their current common ground (Clark, 1996b, p. 67).

Participants in joint actions have a vested interest in posing coordination problems to each other in ways that can be solved. In such *participant coordination problems*, all participants should be able to take it for granted that the one who has posed the problem has not only chosen it and worked out its presentation, but is also anticipating a specific solution and expects that this solution can be readily discerned by everyone involved. This being the case, participants should also be able to assume that the information conveyed in the presentation, in conjunction with their common ground, is sufficient to identify the problem's intended solution. When time is a constraint, they can further assume that the solution will be one that is immediately apparent. These assumptions are respectively referred to as premises of solvability, sufficiency, and immediacy.

Joint actions that arise in language use most commonly take the form of participant coordination problems. Explicit agreements and conventions are good examples of coordination devices that are ideal solutions for these problems because they are straightforward in nature. Agreements preempt the need to work out other solutions, and conventions solve coordination problems that are recurrent.

2.4.1.2 Signaling

At an even more basic if abstract level, in trying to communicate, people must first manage to coordinate what is meant by one person and understood by another. Engineering the more fundamental joint action that achieves this coupling between speaker's meaning and addressee's understanding is a crucial participant coordination problem in all instances of language use. Ideally, what is needed are whole systems of coordination devices to serve as bases for this particular type of joint action, and this is exactly what signaling systems such as conventional languages are. In essence, when a signal is presented, a participant coordination problem is posed: a speaker means something for an addressee to understand. Signals can be devised from any sort of coordination device that is effective. As long as the

the signal is a part of or can be related to what is salient in both the speaker's and the addressee's common ground, each can presuppose that the understanding (the solution) the addressee will come to is what the speaker intended.

In sending and receiving signals, speakers and addressees rely on the existence of *contingency plans* in their common ground. A fundamental part of these contingencies are pairs of rules or heuristics called *signaling doublets* that can be used to pose and decipher participant coordination problems. A speaker, for instance, may know that a hand sign or a word can be used to denote such-and-such when one or the other is presented, and know or expect that his or her addressee knows this too. Correspondingly, when the same hand sign or word is perceived, if the addressee does happen to know that it can denote the same such-and-such and believes that his or her speaker knows this too, then he or she has a basis for deciphering the signal and thus understanding the speaker's meaning.

Users of languages such as English, rely on many forms of conventional signaling doublets including lexical entries, grammatical rules, and various conventions of use and perspective. Such conventions abstract the rules and regularities of a language, but not the wooliness of its actual use. Nonconventional coordination problems inherently arise in all forms of joint action—and hence in language use—and their solutions depend on participants' communicative skills—their ability to reason in terms of joint salience, solvability, sufficiency, and immediacy. Traffickers in meaning and understanding invariably must cope with such irregularities as ambiguity, contextuality, indexicality, and layering, and only through nonconventional means can such participant coordination problems be solved. Indeed, nonconventional coordination devices such as explicit agreements, reliance on perceptual saliences, and the use of precedents all repeatedly arise and succeed as a result of participants' natural resourcefulness, their confidence in their presuppositions about each other's communicative skills, and what is common ground between them.

2.4.2 The Element of Timing

Joint actions are themselves composed of smaller acts of coordination that continuously unfold in time. Although participant actions may be balanced or unbalanced—that is, each person may be doing essentially the same thing or one may be leading while the others follow—the process of coordinating the content of joint actions must itself be coordinated. Timing and cognitive resources are always issues in presenting a coordination problem, in discerning its solution, and in rendering a response. Speakers and addressees must continuously coordinate each other's attention and be able to project the moment when each successive part of the joint action should begin and end. According to this view, joint actions can be thought of as being made up of hierarchies of synchronously coordinated *phases*. Each phase is a segment of joint action with a single functional purpose that is bounded by a jointly executed entry point, body, and exit. People employ various strategies for continuously engineering this synchrony. In joint actions that are rhythmic or periodic, phase exits naturally coincide with phase entries, so a simple cadence strategy can be used to predict phase durations and thus each successive entry point. Phase exits and entries can also be coincident when, as is more often the case, joint actions are aperiodic but contiguous, such as in joint activities like conversation. When this pattern holds, a similar but more general entry strategy can be adopted as long as phase entry points are jointly salient and can easily be projected from the nature of the body of each preceding phase. This same strategy becomes a boundary strategy when entries and exits are not coincident. Particularly in coordinating aperiodic joint actions, participants derive their ability to project a phase's duration from their experience with the substance and character of its body. In conversation, for instance, addressees continuously monitor the intonation of the speaker's voice, the syntax of his or her presentation, and other characteristics of the presentation, in order to project the current phase's duration. It is also a given—a part of each conversant's common ground—that such monitoring processes and the process of understanding take time. Processing is in fact a significant part of the joint action that speakers and addressees simultaneously carry out. Making allowances for processing time goes both ways in all joint actions, and people develop robust heuristics for estimating its demands and for interpreting what it says about each other's capacities and state of mind.

2.5 More on the Nature of Common Ground

What distinguishes information that is common ground for a group of people from any other information they may unwittingly hold in common is their mutual awareness of each other's possession of the first kind of information. In other words, while two or more agents are unaware of each other's knowledge of the same information, that information necessarily lies outside of their common ground. But this state of affairs changes when the agents learn

of each other's possession of the same information. Now they can confidently appeal to their knowledge of what they know together to coordinate joint actions involving this matter, whereas before it would have been presumptuous and possibly counterproductive for either to have attempted to use this particular information as a coordination device.

2.5.1 A Formal Representation of Common Ground

Section 2.3.2 informally characterized the notion of common ground as the "knowledge, beliefs, and suppositions" participants each assume they share about an activity, and Table 4 enumerated how the contents of common ground can be divided into three parts at any moment. These parts are an attempt to conceptualize the essential components of each individual's representation of the context of the moment with regard to his or her counterparts. More to the point, included in the whole of each of these representations is not only the individual's sense of his or her own awareness of whatever the situation might be, but just as importantly, his or her sense of the corresponding awarenesses of any others involved. Very broadly, it is the acquisition of this sense of others' awareness that promotes common ground. In doing anything together, people require an explicit, shared basis to indicate any proposition they take to be a part of their common ground with each other. Ultimately any such basis must arise from one individual's confidence in another's ability to be aware of and use this basis, and vice versa. This insight leads to a formal representation of the nature of common ground Clark denotes as "CG-shared":

Common ground (shared basis)
Proposition *p* is common ground for members of community C if and only if:

1. every member of C has information that basis *b* holds;
2. *b* indicates to every member of C that every member of C has information that *b* holds;
3. *b* indicates to members of C that *p*.

In this representation, the phrase "has information" is intended to encompass a range of synonymous terms such as "believes," "sees," "is aware," or "knows," and the term "community" is used to imply a plurality of members. CG-shared's second condition emphasizes that any basis for an element of common ground must be explicitly known to be shared by all members of the immediate community in question—be it large or small. In a broader sense, when b is taken to be each individual's perceptual awareness, this condition's reflexive character (its self-reference) also succinctly expresses the nature of what it is to be in community with others wherein an individual's awareness includes both an awareness of self and an awareness of an analogous awareness in others.

Although it is possible to represent common ground in another reflexive form that deals only with p and sidesteps the role of a shared basis, the representation made by CG-shared is both more comprehensive and more fundamental. A mutually held proposition p' can easily be supported by inequivalent, unshared bases b_1 and b_2, but the correct characterization of this circumstance is not that p' is nevertheless common ground but that b_1 and b_2 in not being shared do not justify p' as common ground. Experience teaches that holding different bases encourages misunderstandings, hence the following *principle of justification*: In practice, people take a proposition to be common ground in a community only when they believe they have a proper shared basis for the proposition in that community (Clark, 1996b, p. 96).

What is individually taken to be common ground, though, is an inherently subjective matter. Individuals must evaluate every potential shared basis in terms of both the common ground it indicates and its *quality of evidence*—how well it justifies a given proposition. According to the *principle of shared bases*, for something to be a coordination device, it must be a shared basis for a piece of common ground (Clark, 1996b, p. 99). Thus, any supposedly shared basis that is poor evidence for a difficult or dubious inference will likely make a weak coordination device because it has little likelihood of indicating the intended solution to others. To succeed in coordinating their joint actions, individuals have a vested interest in striving to find or contrive the most robust shared bases possible.

2.5.2 Finding and Building Common Ground

Shared bases are so fundamental to the broad conduct of cultural activities that entire communities of individuals can be said to be defined around them. By consensus, members of such *cultural communities* can assume that they

share what is called *inside information*, a richer knowledge of their community's concerns than would be possessed by a nonmember. Their inside knowledge is a form of shared expertise that is *outside information* for others. Membership is a powerful shared basis. People display their own and look for evidence of others' membership in the various cultural communities to which they belong because their joint interests and agendas are so greatly facilitated when they can establish this *communal common ground* with others.

Many different forms of information are characteristic of communal common ground. People's shared knowledge of their basic human nature and the properties of the natural world in which they live are perhaps most fundamental. More definitive are communal lexicons, histories and facts, and other cultural conventions such as a community's norms and procedures. In addition to this are a community's unique perspectives and experiences, many ineffable, that constitute it's most exclusive inside information. Another characteristic associated with communal common ground is people's generally robust ability to accurately grade the degree to which they share information with others based on such cues as their awareness of how one community is related to another and their personal intuitions and assessments of their own and each other's informational capacities.

In addition to communal common ground, people rely on a broad class of shared bases that arise from their personal experiences with each other. For the most part, the experiences that make up an individual's *personal common ground* with others are either naturally occurring *joint perceptual experiences* or joint actions. A shared perceptual basis is anything jointly perceived *as it is*, whose salience may be indicated by its obviousness or by observing another's attention or gesture. Ordinarily, any such instance of *perceptual copresence*—witnessing the same thing together—satisfies each of the conditions of CG-shared. Perceptual bases are said to have *natural meaning* because they are not contrived but are instead naturally occurring. What a joint perceptual experience is specifically taken to mean, however, generally depends on the communal common ground of the individuals perceiving it.

Inferring personal common ground from the shared basis of a joint action also depends on the communal common ground of the individual participants. In contrast to perceptual experiences, though, joint actions are said to have *non-natural meaning* because they are intentional events and rely on signaling in order to be carried out. Once the signal meaning of a joint action has been conveyed and understood, its significance can be taken to be common ground among the participants. Engineering such events so that they too satisfy the conditions of CG-shared is a fundamental process in using language.

Shared bases for personal common ground are autobiographical in nature and often include personal lexicons. They also play a defining role in determining who is a friend, an acquaintance, or an intimate, and who is a stranger—generally, the greater the personal common ground between two people, the greater the acquaintance.

Indeed, any degree of common ground that may exist between people must be established and built up in their encounters with each other. Various forms of circumstantial and episodic evidence become the shared bases for inferences of common ground. Such evidence can be intentionally or unintentionally disclosed, but it is necessary for this to be explicitly recognized by each of the parties involved for it to establish any facet of their common ground. Ultimately, common ground develops in layers. Each new stratum always relies in some way on an earlier piece.

2.6 Other Fundamental Notions in Language Use

In the preceding sections, many of the broad fundamentals of Clark's characterization of language use have been set out. People engage in joint activities in order to do things together, and in pursuit of their social ends, they inevitably must find ways to communicate with each other. Language, in its broadest sense, solves this problem by enabling one person to convey his or her intention to another and achieve its recognition. To carry this out, people participate in acts of jointly coordinated reasoning that take advantage of the fact that individuals possess reasonably similar conceptions of the world in which they live. This particular kind of joint action is essential to people's joint activities. It is not only how they get others to understand what they mean but also how they come to know what they have in common—how they establish their common ground. In joint actions, people pose their intentions to each other by proffering elements of their common ground as a signal basis for recognizing what those intentions are. It is then incumbent on audiences to make these connections, and their willingness to coordinate with their counterparts in this way is what makes communicative actions joint actions. As these interactive experiences accumulate, so do people's

knowledge of each other, thereby enlarging their common ground—the shared basis that makes their joint endeavors possible. Still, there are a few additional points that should be made, particularly in regard to joint actions. For Clark, this participatory process is central in language use.

2.6.1 Speaker's Meaning and Levels of Action

On closer inspection, the notion and role of meaning in the use of language is a bit more subtle than might be expected. Meaning in the sense of one person trying to indicate something for another to understand is fundamentally different from the intrinsic or symptomatic meaning everyday things in the world have by virtue of their inherent nature. This distinction, which is due to Grice (1957), divides meaning into two types (mentioned earlier in Section 2.5.2)—respectively, *non-natural meaning* and *natural meaning*. Both come into play in different ways in joint actions and in language. Because natural meaning is, in effect, self evident, its primary role in language use is commonly that of a tacitly recognized element in the common ground of the moment in people's joint activities, though it may also play other roles. Non-natural meaning, in contrast, is distinguished by its association with the notion of signaling. Its role is central in joint actions.

Non-natural meaning itself has two components—*speaker's meaning* and *signal meaning*. The difference between the two is the difference between what a person intends to be understood when he or she presents a signal and what that signal may mean per se. Speaker's meaning is inherently precedential because signal meaning can only arise from use. But speaker's meaning can only be conveyed when an effort is made to understand it. That is, the meaning intended by a speaker can only be recognized when the intention that it be recognized is itself recognized by addressees who then must do their part. The principle is that *communicative acts* are inherently participatory and necessarily entail both *signaling* and *recognizing*. Clark's formal representation of the joint nature of speaker's meaning reflects this:

Speaker's meaning (joint)
In presenting signal *s* to audience A, speaker S means for A that proposition *p* if and only if:
0. the communicative act *r* includes 1 and 2;
1. S presents *s* to A intending that *p* as part of *r*;
2. A recognizes that *p* as part of *r*.

The fact that speakers, in signaling, intend for their audiences to perform a corresponding act of recognition means that in communicative acts, speakers are making a kind of proposal. Very often, in fact, they specifically want their addressees to do something more—and seemingly, they are trying to accomplish all of this through the mere action of their presentations or *utterances*. Such utterances, linguistic or otherwise, are called *speech acts* (Austin, 1962). If a speaker can get a listener to recognize his or her meaning, the effect is called an *illocution*. If through this understanding a speaker can get the listener to take some further action, even if unintended, the effect is a *perlocution*. An Illocution may stand on its own but perlocutions are always accompanied by illocutions, and what these terms describe is the discharge of speaker's meaning in one form or another. Hence, and more to the point, illocutionary and perlocutionary acts cannot be accomplished without specific cooperative responses from the listener—responses that complete specific joint actions. Speakers are in effect beholden to their addressees to complete what they are proposing.

As it turns out, speakers and addressees must do quite a bit together to carry out a communicative act. What at first appears to be one joint action is actually a hierarchy of several essentially contemporaneous joint actions. This is hinted at in the way perlocutions depend on illocutions. Such hierarchies are termed *action ladders* because they can be decomposed into series of causally ordered, metaphorically ascending levels of action. Action ladders are ubiquitous in daily life, and speakers and addressees carry out prototypical examples. A speaker or signaler must *execute* a sign in order to *present* the sign in order to *signal* with the sign in order to *propose* one thing or another. Similarly, an addressee or a recognizer must *attend* to a sign in order to *identify* the sign in order to *recognize* the sign's meaning in order to *consider* responding. The *upward causality* that is seen in such action ladders defines an ordering that is irreflexive, asymmetric, and transitive. Two properties follow from this that people intuitively depend on and use. The first is the *property of upward completion*: In a ladder of actions, it is only possible to complete actions from the bottom level up through any level in the ladder (Clark, 1996b, p. 147). The other is the *property of*

downward evidence: In a ladder of actions, evidence that one level is complete is also evidence that all levels below it are complete (Clark, 1996b, p. 148). When the actions that signalers and recognizers carry out at each level in their respective action ladders are taken for what they properly are—pairwise parts of specific, order-dependent joint actions in a communicative act—an action ladder of joint actions emerges that exhibits the same upward causality and the properties of upward completion and downward evidence:

Level4 Proposal and consideration
Level3 Signaling and recognizing, or meaning and understanding
Level2 Presenting and identifying
Level1 Executing and attending

Reprinted (with modifications) from Clark, *Using language*, p. 153, ©1996 Cambridge University Press

2.6.2 Proposing and Taking up Joint Projects

Level 4 in the ladder of joint actions described above is something that goes beyond meaning and understanding. It is the entertainment of what Clark calls a *joint project*—a joint action projected by one of its participants and taken up by the others (Clark, 1996b, p. 191). The object of this top-level joint action is to move the participants' real concerns—their *official business*—a step further along. But for an addressee to complete the joint action begun at level 4, he or she needs to have already completed the transaction at level 3. That is, the addressee must already have understood or construed enough of what the speaker's intentions are to formulate a viable response. Although full construals are not always possible or easy to make, in responding, an addressee both agrees in some way to take up a joint project and comes forth with evidence of the measure of his or her understanding. In the event of a misconstrual, the addressee may need the speaker's help. To be sure, construals are a problem for both parties in a joint project. Together, they must settle on what the speaker means, and this is called the *joint construal problem*.

Construals are fundamental to making sense of things. From the most basic inferences of natural meaning to the most difficult exercises of comprehension, construals account for most of what people understand—and do. Indeed, people often signal their construals publicly in order to display to others the nature of their attitudes, their conclusions, and what they understand in any number of social contexts. And so it is with speakers and addressees in their communicative acts and joint projects. Signaling actions that are meant for others are often met by or paired with responses that are intended in part to indicate construals. Such responses are usually open to evaluation. Construals can be right or wrong or somewhere in between; they may be nominally flawed or their flaws may go undetected. How a speaker subsequently intervenes or moves on helps to settle the joint construal problem that proceeds from his or her initial utterance.

The *action-response pair* pattern just described is indispensable in joint activities. Joint construals can only be achieved by comparing notes, as it were, and from this, such turn-taking naturally emerges. Just as important, though, are the larger, moment-to-moment concerns of the joint activity itself, the joint projects carried out at level 4 that presuppose joint construal at level 3. Proposals and their consideration and uptake also require two-part exchanges, and when action-response pairs are employed for joint projects, they have the following properties and are characterized as *adjacency pairs*:

1. Adjacency pairs consist of two ordered actions—a first part and a second part.
2. The two parts are performed by different agents A and B.
3. The form and content of the second part is intended, among other things, to display B's construal of the first part for A.
4. The first part projects uptake of a joint task by the second part.

Reprinted (with modifications) from Clark, *Using language*, p. 201, ©1996 Cambridge University Press

Adjacency pairs efficiently solve two of the problems participants in joint projects face at once—how to conduct business (property 4) and how to settle on what they are doing (property 3). In particular, by the property of downward evidence in the ladder of joint actions in communicative acts, an addressee's uptake can be taken as evidence of his or her understanding. As the smallest of possible joint projects, the proposal and uptake pattern of adjacency pairs is ideal for carrying out joint actions with a minimum of joint effort. It establishes who the participants are and

implicitly defines their roles, it helps the participants to coordinate the entry times of their joint actions' phases, and it standardizes a means for jointly introducing, construing, and otherwise acting on the content of their joint actions—their official business.

Finally, there is the role of purpose to be considered in joint projects. The purpose of any joint project is to accomplish something, but what that something is must be jointly worked out by its participants who must then demonstrate their commitment with their actions. In particular, there are these requirements:

For A and B to commit themselves to joint purpose *r*

1. *Identification* A and B must identify *r*
2. *Ability* It must be possible for A and B to do their parts in fulfilling *r*
3. *Willingness* A and B must be willing to do their parts in fulfilling *r*
4. *Mutual belief* A and B must each believe that 1, 2, 3, and 4 are part of their common ground

Reprinted (with modifications) from Clark, *Using language*, p. 203, ©1996 Cambridge University Press

Somewhat like joint construals, joint purposes must also be settled on if these requirements are to be met. When a respondent fails to proceed or agree to a mutual purpose in full or in part, he or she is invariably unable or unwilling to do so. Because all of these requirements are subject to negotiation, each participant has a role in determining what joint purpose a joint project will serve, whether the project is minimal, in the case of adjacency pairs, or extended. Indeed, extended joint projects are just the emergent enterprises of lesser joint projects opportunistically engineered to serve larger purposes though chaining, embedding, and other tactics. In particular, what actually transpires in most joint activities is as much a bending of agendas as it is anything else. Clark notes that in reaching joint construals, what the speaker means is principally replaced by what the participants mutually agree to take the speaker as meaning. An uptake will often signal which of several construals an addressee is willing to entertain. And so individual agendas and joint purposes change and adapt.

2.7 Elements of Language Use in Human-Computer Interaction

In its essence, Clark's theory of language use is a theory of how people solve the problem of doing things with each other. Beginning with their earliest face-to-face encounters, people acquire and refine a complex but consistent body of communicative skills that becomes one of their greatest assets in life, serving them in one capacity or another in virtually every setting in which language use plays a role. Until relatively recently, it could also be taken for granted that these skills were uniquely reserved for doing things with people.* However, with the rise of technologically sophisticated information processing machinery in this century, this assumption has been challenged—at least in practical terms. Increasingly, people are entering into activities with machines that have much of the character of their joint activities with people. While it is correct to say that people autonomously use tools, machines, and machine-based systems because they have been designed to amplify or facilitate human limitations or implement codified processes, etc., it is nevertheless the case that many human-computer interaction events supported by user interfaces are, in effect, communicative acts between users and computers. That is, many systems are designed not just to interact reflexively with users, but to work at times with their users in ways that strongly resemble the manner in which a "real agent" would work.

This last point was raised by Clark in his plenary address to the Association for Computing Machinery's 1996 Conference on Human Factors in Computing Systems (CHI '96) in Vancouver, Canada. In the short paper that accompanied his talk (Clark, 1996a),† he argued that minimal joint projects between people (i.e., adjacency pairs) are of interest in the design of human-computer interaction because "they have long been the model for communication with and through machines." People have "systematic, economical, and robust" ways of solving the difficulties that arise in arranging to do things with each other, and these techniques are employed to coordinate even the smallest of joint actions that advance their larger joint activities. In analogous joint activities with computers and other interactive technologies, people intuitively try to adapt these same skills to advance their machine-supported tasks. Hence, an understanding of the underlying principles at work in people's socially based communicative acts is needed if their corresponding skills are to be successfully supported in the design of interactive systems.

*To some extent, companion animals could arguably be included here, too.

† All quoted material in this section is from Clark (1996a).

To support the argument that interaction designs regularly anticipate people's communicative skills, Clark provides a number of common examples in which a proposal plus its uptake—what defines a minimal joint project—is used as the paradigm for implementing transactions between users and machines and vice versa. He is careful to point out that such instances are only genuine joint projects when the technology is used as a medium for conveying one person's intent to another, such as in ringing a telephone. In contrast, joint projects that involve machine-initiated proposals or machine-driven responses only have "the look and feel" of the real thing. These designs succeed, however, because users "know what to do…by analogy with genuine joint projects." Indeed, the chief rationale for a range of interaction techniques supported by graphical user interfaces is that they too are self-evident, or are quickly learned, by analogy with direct manipulations people carry out in the real world. However, when interfaces reflexively and/or metaphorically represent user inputs as autonomous actions that appear to be entirely under the user's control, such events would not seem to qualify as joint projects, in spite of the machine's invisible support of the metaphor. Rather, joint projects in machine interaction designs have the functional character of adjacency pairs, and when user interfaces fail to adequately leverage or coordinate any of the terms people ordinarily try to establish before they enter into joint actions, the user experience is certain to be less than optimal.

The points that Clark raises concerning the applicability of principles of language use to human-computer interaction have implications that go well beyond the introductory examples he provides. Cognition plays a role in virtually every joint action people enter into, and many of its underlying processes are poorly understood and/or simulated or are entirely absent in current interaction designs. As Clark notes, "It takes delicate coordination against the common ground of the participants to initiate [joint] actions." This is particularly true in the conduct of extended joint projects. When these cognitive skills are effectively unavailable, the range of expression and common ground in the conduct of joint activities can be severely limited. If user interfaces are to achieve greater fluency in this regard, it will be necessary for researchers, technologists, and designers to identify and deploy machine-based methodologies that are capable of simulating the cognitive skills people regularly employ in their face-to-face projects with each other, as well as in other settings in which language use plays a role. In the next section of this report, a modeling effort with this agenda is described.

3. TASK MODEL TRACING

3.1 Introduction

Common ground entails both knowledge and process. When people and computers come to a task, they both begin with a certain amount of background information. For people, this is a body of presuppositions they have about a number of things, such as the task, the computer, their role as a computer user, and the task's real-world context. Altogether, this is a representation that serves as the basis for how they will proceed. In principle, the situation is much the same for computers. The structure, design, and data underlying an application and its user interface embody a design team's presuppositions about many of the same issues—the task, the computer, the user, etc. And this too is a representation; it stands on its own and determines how the application software will proceed. These two complementary representations are necessary if users and computers are to do things together. And the ways in which they correspond and overlap constitute the user's and the program's initial common ground.

The knowledge component of common ground is a picture of an activity's state of affairs—what each of its participants presupposes is jointly known about each other and what they are doing together at the moment. The process component, in contrast, determines how the knowledge component is shaped. As the joint activity of human-computer interaction advances, each participant's knowledge changes through a process of accumulation. Users ruminate and keep track of the computer's actions and their own, and this accumulating information is steadily added to their representation of the activity. Programs also change their representation of things as activities advance. Their displays change and internal states change and their stored data are modified. For the program, many of these changes are informed by what are expected to be the user's needs—information about the program's part in the activity that the user will likely want to be able to get. What programs and users jointly presuppose about these accumulative processes in each other is just as much a part of their common ground as the other presuppositions they have in common.

Many of the changes that occur in human-computer interaction are evanescent, leaving little or no trace of their occurrence in the environment. There are, of course, notable exceptions to this. Computers are used for—and excel

at—many forms of record-making activities. But in most cases, these exceptions occur as the focus of an activity rather than in the doing of it. Evanescence is a property of most joint activities (see Table 1), and humans have evolved a sophisticated cognitive resource for coping with it, namely their memory.

People use their memory to maintain a cumulative record of their understanding and perceptions of an activity's events. In joint activities, the events that are public are usually taken to be part of the participants' common ground—a record they presumably share of the activity's public events so far (see Table 4). But there is more to it than this. People's internal record of an activity's events so far is usually rich with inferences, annotations, interpretation, intention, and meaning, all of which are products of their cognition as they participate in an activity. Between people, these features of cognition are taken for granted; among other things, people rely on each other to think about what they are doing and to recognize evidence of thinking in each other. In their joint activities with computers though, these cognitive aspects of common ground can be, and usually are, broken in several ways.

History mechanisms in conventional user interfaces, for instance, are not maintained by programs for their own use. Instead, they exist as a tool for users who may wish to undo or repeat an action or see an explicit record of what they have done. Invariably too, the actions recorded are only a portion of what users might reasonably expect to be a record of the activity's public events so far. Even so, since programs do not ordinarily make internal use of the history they do maintain, users cannot rely on the bulk of their own analysis of these seemingly conspicuous events to be shared by programs for the mutual purpose of advancing the task—their joint activity. This is just one example. More often than not, programs' actions are simply event-driven—evanescent, reflexive reactions to their users' inputs. Their behavior is effectively unconsidered, and users inevitably must accommodate this handicap in the common ground of their joint activities with computers.

The situation between users and computers—the deficiencies in their common ground—is not likely to change appreciably until programs are more consistently designed to simulate the cognitive skills people bring to their participation in joint activities. How this is done may not be as important as whether or not it is done in the first place. Potentially, any number of computational strategies can be employed in user interfaces to simulate cognition to some level of fidelity—arguably, any methodology used to achieve an "intelligent" user interface qualifies. Simulated cognition in user interfaces adds to the potential common ground in programs' activities with users. But sophisticated capacities are also merited only to the extent that they make genuine contributions to interaction designs.

In one sense, using software is somewhat like reading a book. The user-reader and the software's designer-author engage in a joint activity whose communicative medium is spelled out in the material design of the software's user interface. Design concerns in this perspective are essentially issues of common ground that are relevant in what amounts to a written or preproduced setting (see Section 2.2). However, in a broader and more important sense, the interactive experience itself is more like a joint activity *with the software itself* in a face-to-face or copresent setting. Another way to understand the nature of this dual interpretation of an interaction design is to state it in terms of what Clark calls *layers of action* (also discussed in Section 2.2). In this view, the user and the software's designer carry out a multilayered joint activity through the medium of the software's user interface. In the primary layer of action, common ground is established between the user and the designer through the medium of the design of the user interface. In the secondary layer, which transcends the literal circumstances, common ground is established between the user and the running program. In other words, it is the intention of an interaction design for its user to interact with the user interface—not with the designer—and to adopt the world of its particular representation of the application task. In this sense, the more a program's interactive capacities appear to have a cognitive basis, the more the line between software as medium and software as agent blurs, hence the bias of this report—that users and computers are *both* participants in the joint activity of human-computer interaction. If this bias is taken seriously, then the notion of common ground itself should be recognized as the single most important organizing principle in the design of interaction. In any event, it is arguably paramount in the design of simulated cognition for interactive purposes.

3.2 Simulating Cognition in a User Interface

In this and the following sections, I describe an effort to simulate an interactive cognitive function in an application user interface. The aim is to explore how, as a participant in the joint activity of its application task, a user interface can more consistently support the accumulation and use of task-related common ground with its user.

There are several working premises in this effort. Since common ground is fundamentally a cognitive process, simulating its computational nature merits an approach through cognitive modeling. This leads to the notion of a *task model*, which functions as a simulated cognitive representation of the task from the application's point of view. Part of this representation must include a representation of what the user knows about the task, which is sometimes characterized as a *user model*. In the context of this work, the user model more specifically corresponds to a representation of potential and established common ground with the user. Just as people have personal and shared representations of their joint activities with others, the full task model can be thought of as the user interface's *personal* representation of the task, and its user model as its *shared* representation of the task (compare with the material at the end of Section 2.3.2).

Cognitive modeling in this effort requires an application task analysis that not only addresses representing the task as just described, but also takes into account any interactive cognitive functions envisioned for the user interface. Since common ground between two agents must be established interactively (Section 2.5.2), at a minimum, the task model must be designed to interpret task-related events and to share its knowledge of the task with the user. Thus, in the work presented here, interactions between the user and the application are treated as joint actions of communication as described by Clark (see generally Sections 2.4 and 2.6.2, as well as the discussion of *action ladders* at the end of Section 2.6.1). In particular, participants in joint actions involving meaning and understanding carry out subordinate joint actions in cognition—the intention behind a signal is formed cognitively by the one presenting the signal, and the process of recognizing and construing that intention is carried out cognitively by the audience for the signal. As a participant in carrying out the application task, the user interface, through its task model, must be able to simulate these specific cognitive functions in its joint actions with the user.

As suggested in Section 3.1, making a pronouncement on how a set of task-related cognitive skills should be simulated in a user interface (i.e., what specific methodology or technique should be employed for this) is not part of the agenda in this effort—ultimately any computational technique that works well can arguably justify its use. The chief rationale for using cognitive modeling to simulate cognition in the work presented here is to approach an ostensibly cognitive process—common ground—from the principled perspective of a theory of cognition. The material that follows covers the modeling and implementation issues faced in this effort and presents a functional picture of how the resulting system works. Specifically, a description of how a small application prototyped in Common Lisp has been modified to work with a particular cognitive modeling environment, and how the task model's function is conducted with respect to the actions and needs of the user and the application is given. Throughout, references are made to the elements of Clark's theory of language use as joint action the modeling work strives to address. In particular, the work is concerned with cognitively modeling elements of the application task that have the potential to be established as common ground between the application and the user, and can be used by the simulated cognitive function in the user interface to aid the user in understanding and carrying out the task itself. Only a portion of the application task has been modeled sufficiently enough to credibly demonstrate the stated goals of this effort, but two examples of how common ground is used by the system are discussed in detail in Section 3.5.

The working system is called a *task model tracing* system because cognition is simulated by "tracing" application events in the task model. No attempt is made to model the use of natural language. Instead, the interaction medium—functionally, the language used by the user and the computer—is the standard point-and-click paradigm supported by graphical user interfaces, in which users communicate predominately with a mouse and a keyboard and the interface communicates predominately with actions, options, and statements in its display. In particular, the task model interactively shares its knowledge of the task with the user through reports, composed of predetermined sentences, and similarly predetermined advisories that are presented in the form of checkbox lists. All of this material is displayed in a special window designated for this purpose; how and when the material is chosen for display is a function of the task model.

Since the task model tracing system represents an effort to simulate cognition in a user interface for purposes of human-computer interaction, the focus in the work presented here is not on issues of common ground in the design of a conventional user interface, but on the problem of computationally modeling common ground between the application and the user as an accumulative process (see Section 1.2).

The task model tracing system comprises four components: the host application, the cognitive modeling environment (a computational simulation of the human cognitive architecture), the task model, and some additional constructs needed to round out the system. The general relationship between these components is shown in Fig. 1. The host application implements an information environment for carrying out an imaginary planning task. The cognitive modeling environment can be thought of as a processor for the task model, which itself can be thought of as a kind of cognitive program. The remaining component groups together all of the modifications and additions to the system and the user interface that permit the user to interact in a direct manner with the task model, and provide the task model and the application processes with access to each other. To appreciate how the components of the task model tracing work together, it is first necessary to understand the basic features of the cognitive modeling environment and the implications this environment has for the design of the task model.

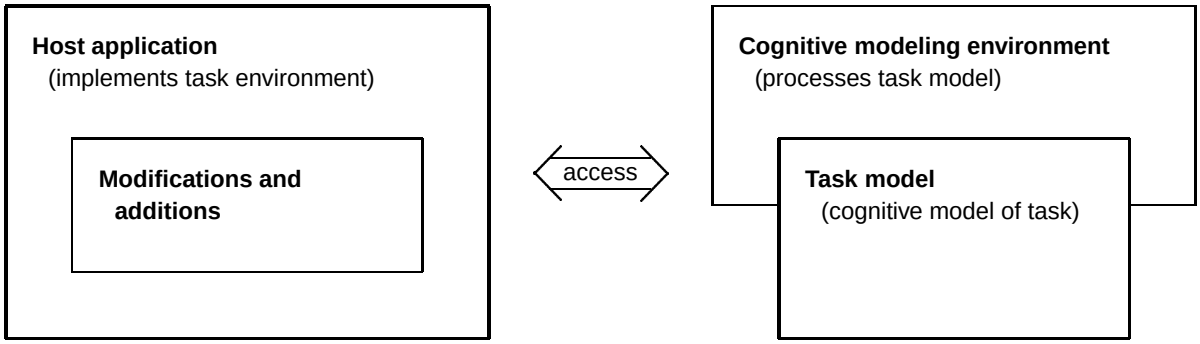

Fig. 1 — The four components of the task model tracing system

3.3 ACT-R

ACT-R (Anderson and Lebiere, 1998) is one of a small number of *unified theories of cognition* that have emerged in recent years as a result of what is sometimes called the "information processing revolution" in the cognitive sciences (Simon and Kaplan, 1989). Also called *cognitive architectures*, unified theories of cognition are more or less complete proposals about the structure of cognition that strive to account for the full range of cognitive behavior with a single, coherent set of mechanisms (Newell, 1990; Anderson, 1993). Computer implementations of cognitive architectures provide what are effectively structured programming languages for simulating and modeling cognition. ACT-R has been used to provide compelling accounts of high-level cognition in a wide variety of circumstances including navigation, scientific discovery, language comprehension, and problem solving (Anderson and Lebiere, 1998). A central claim of the theory is that cognitive skills are realized by production rules—*if-then* constructs that account for many of the procedural characteristics of cognition (Anderson, 1993)—and as such, ACT-R is an instance of a more general computational framework known as a *production system*. Production system architectures have emerged as a dominant tool for complex cognitive performance modeling (Klahr et al., 1987) and have proven to be well suited for cognitive research in human-computer interaction.

ACT-R differs from other production system-based theories of cognition in a number of ways that make it a good choice for studying certain aspects of common ground in a cognitively augmented human-computer interaction environment. Among these differences are the way it views knowledge and the control of attention, and the way it accounts for the performance characteristics of memory. Conceptually, the theory functions at two levels, one *symbolic* and the other *subsymbolic*. Information is processed at the symbolic level in the sense of how it is acquired, used, and stored. At the subsymbolic level, information is processed in neural-like terms using continuously varying quantities that determine its availability and speed of access, among other things. Since cognitive modeling in ACT-R takes both of these processing levels into account, a sketch of each, relevant to the work in this report, is given in the next two sections.

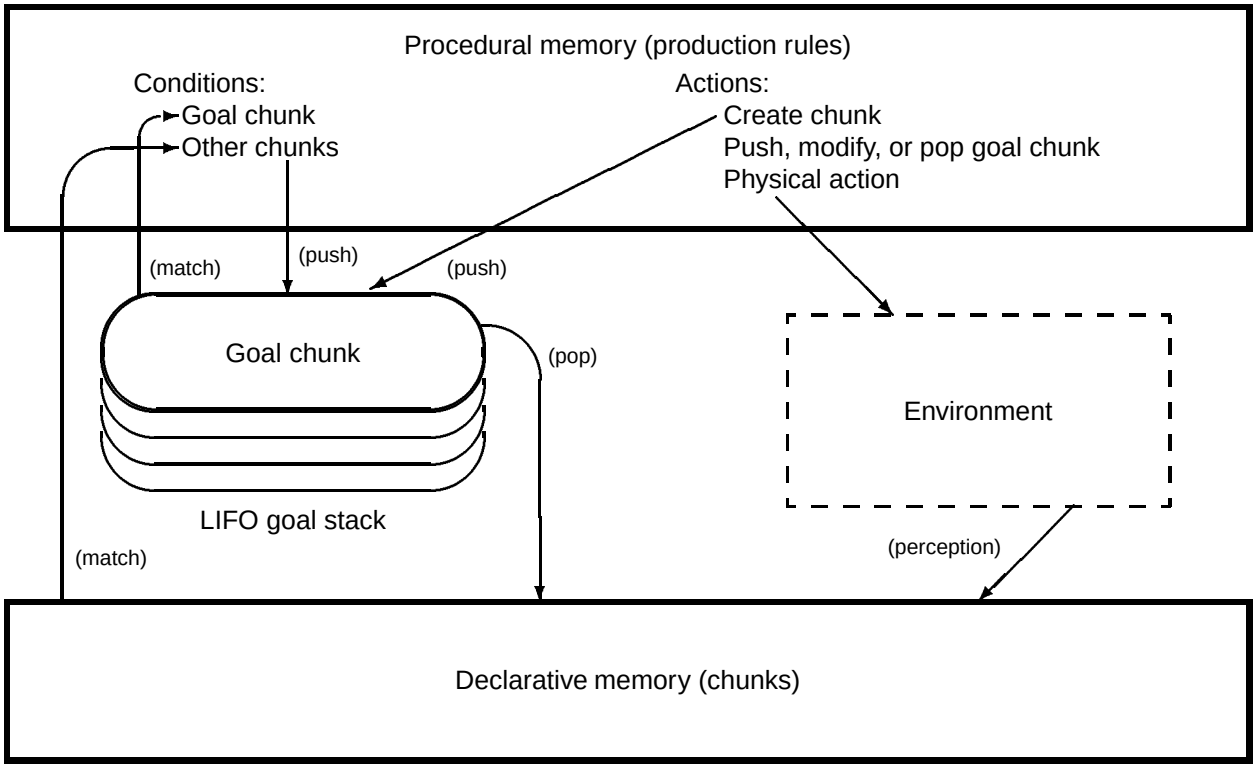

Fig. 2 — A diagram of most of the features of the ACT-R architecture at the symbolic level

3.3.1 Procedural and Declarative Knowledge

Figure 2 shows that ACT-R divides knowledge into two kinds of symbolic memory—*declarative* and *procedural*. Declarative knowledge is essentially the kind of conceptual or factual knowledge people are aware of knowing and are usually able to recall. In ACT-R , declarative knowledge is represented in a form known as *chunks*, which can be thought of as independent, coherent nodes of information. Chunks are stored in a "long-term" *declarative memory* and are modeled notationally as patterns of *slots* that are assigned values that are themselves other chunks, for the most part. A given, arbitrary pattern of slots defines a generic chunk type, and instances specify this in a special reserved slot known as an *isa* slot. As an example, the user, from the computer's point of view in an ACT-R model of a segment of human-computer interaction, might be represented as an instance of a "participant" chunk as follows:

user
    isa participant
    label "user"
    attend no

The chunk is given an arbitrary name, in this case the word *user*, and its type, *participant*, is shown in its *isa* slot. This chunk type happens to have two other slots, a *label* slot and an *attend* slot, and their values in this instance are also shown in the example.

In ACT-R, chunks are created, stored, modified, and retrieved through the action of *production rules*. Production rules (or simply "productions") are symbolic representations of procedural knowledge—essentially, behavioral skills—that are stored in a separate long-term *procedural memory*. They are called "rules" because each production pairs a set of *conditions* with a set of *actions*:

if
    C (a set of conditions) is true
then
    carry out **A** (a set of actions)

A rule's conditions, which are simply a template, are always expressed in terms of a privileged chunk, known as the *goal*, and other chunks in declarative memory that may be related in some way. ACT-R orchestrates the focus of attention in cognition through a last-in-first-out (LIFO) stack mechanism that is managed by sequences of production rule actions. The architecture operates in *cycles.* To become the current goal, a chunk must be created, or retrieved from declarative memory by a production as one of its conditions, and then "pushed" onto the stack. At the start of each cycle, ACT-R first chooses a production that matches the current goal and then tries to match the production's other conditions against the chunks in declarative memory. This process, called *conflict resolution*, is described in more detail in Section 3.3.2. As soon as a corresponding set of matching chunks is found, they are retrieved and *bound* to the production, and the instantiated result is "fired" (its actions are carried out); this finishes the cycle.

Several of the actions a production can take are shown in Fig. 2. In addition to creating chunks and selecting goals by pushing chunks onto the stack, productions can also modify and remove (or "pop") the current goal. When a chunk is popped off of the goal stack, ACT-R returns it to declarative memory. Cognitive models must also be able to interact with their environment, and ACT-R handles this by allowing the modeler to add chunks directly to declarative memory (simulating perception) and by allowing productions to simulate "physical" actions through Lisp function calls.

ACT-R's distinction between declarative and procedural representations of knowledge fits well with the general characterization of common ground as both knowledge and process emphasized in this report (Sections 1.1 and 3.1). The knowledge component of common ground—information that is presumed to be shared by the participants in a joint activity—corresponds to a declarative representation in ACT-R. The process component of common ground—the cognitive skills participants employ to accumulate and use the knowledge component—corresponds to a procedural representation in ACT-R. For instance, in Clark's characterization of the three parts of common ground in a joint activity at any moment—*initial common ground*, the *current state of the joint activity*, and the *public events so far* (Section 2.3.2)—each part can be represented as a body of declarative knowledge, in the form of chunks, that is shared by the participants. The cognitive skills that each participant employs from moment to moment to update, maintain, and use this body of shared declarative knowledge he or she possesses can be represented in procedural form as a set of production rules.

A similar correspondence can be seen in the terms Clark uses to formally describe the nature of common ground (Section 2.5.1). Here, both the notion of a *shared basis* and the notion of a *proposition* that is common ground for the members of a community are well suited to being represented declaratively as chunks. The role of *indication*—the idea that a shared basis *indicates* a proposition to all of the members of the community)—is ideally represented procedurally as a production rule. ACT-R's goal stack facilitates the relationship between these two representational forms of knowledge. When a declarative, shared basis is salient and in focus for the participants in a joint action, it is their procedural knowledge that carries out the cognitive process of inference that indicates the declarative proposition taken to be in their common ground. In ACT-R, this relationship between knowledge and process in common ground can be modeled by pushing a chunk corresponding to a shared basis on the stack, making this shared basis the current focus, or goal. Production rules that match this goal then represent processes of inference that lead to propositional elements of common ground. Both declarative knowledge and procedural representations of cognitive skill are needed to provide a computational account of common ground in ACT-R .

3.3.2 Rational Analysis and Subsymbolic Processing in ACT-R

Although the acronym *ACT* has a number of variants, the *R* in ACT-R stands for Anderson's *rational analysis* of cognition, a theory that takes cognition's adaptive capacities to be an optimized evolutionary response to the statistical character of the environment (Anderson, 1990). Using Bayesian estimation techniques, correspondences between the environment's demands and performance in a wide variety of cognitive phenomena, including recall, categorization, causal inference, and problem solving, can be understood as rational behavioral adaptations when analyzed in terms of cost vs benefit. The insights of this theory and its use of Bayesian methods are the basis of ACT-R's account of how

the cognitive architecture functionally processes information at its subsymbolic level. There, chunks and productions are each subject to continuously varying levels of activation, which reflect estimations of their importance based on previous use.

The consequences of subsymbolic processing in ACT-R are expressed at the symbolic level in the process of conflict resolution, mentioned previously in Section 3.3.1. At the beginning of each cycle, any number of productions may happen to match the current goal, but in ACT-R they are ordered in terms of their *expected gain*. This ordering is called a *conflict set*. Each production's expected gain is a measure of how likely the production is to be useful in completing the activity represented by the goal chunk, based on past experience and an estimate of how much effort the activity will take to complete. The first production in the conflict set gets a chance to match the remainder of its conditions, but if this fails for some reason, the next production is immediately considered and so on, until the process is resolved. Each production's success or failure in the conflict resolution process is then taken into account when ACT-R begins the next cycle.

The process of matching a production's nongoal conditions during conflict resolution is also subject to a probabilistic measure—the activation levels of candidate matches in declarative memory. ACT-R's strategy for assigning these values reflects the circumstances of each chunk's current and past use. Chunk activation values serve two purposes in conflict resolution—to order chunks so that the most active match for each condition is chosen and to determine the effort needed to retrieve the matches, which is used in the estimation of expected gain.

Although the specific details of ACT-R's subsymbolic computations are different for production rules and chunks, at a grosser level, they have certain characteristics in common. A central feature of the theory is that these quantities are "learned" (increased) through use and are subject to decay over time. In particular, learned chunk activation values, which reflect frequency and recency of use, can be used to represent the *salience* of declarative representations in memory. Salience plays an important role in joint actions—it enables the participants in joint actions to find solutions in their common ground to the problem of coordinating each of the subordinate actions that contribute to their joint acts of communication (see Clark's "principle of joint salience" in Section 2.4.1; see also Section 2.6.1). An example of how this subsymbolic measure of salience can be used to indicate a proposition in common ground is given in Section 3.5.

To work (i.e., to run), ACT-R models must be processed by the ACT-R simulation of the human cognitive architecture—much like computer programs require particular operating systems and processors to run. ACT-R's cognitive modeling environment can be configured to work with or without most of the components of the theory's subsymbolic processing. The advantage of this arrangement is that cognitive models can be developed in stages or can be organized to address specific considerations. In part, both of these approaches have been taken in developing the model presented in this part of the report (i.e., Section 3); however, not all of ACT-R's subsymbolic processing mechanisms have been utilized. The aim has not been to create a highly plausible cognitive model but to appreciate a number of computational problems user interfaces face in simulating the cognitive demands of common ground in interactions with users.

3.4 The Task Model Tracing System

This section takes up the considerations raised in prototyping a system in which a limited set of task-related cognitive skills is simulated in an application user interface with an ACT-R model. The widespread complaint that computers lack ordinary cognitive skills suggests that users recognize an inherent psychological and social dimension in the proposition of human-computer interaction (compare with the notion of "social interaction of machines" in Norman, 1992). The current work is motivated in part with this in mind, and by the conjecture that if users want applications to be able to reason as they do, perhaps fundamental cognitive principles should underlie the computations.

There are certain practical shortcomings to an approach through cognitive modeling. For instance, like most other production system architectures, ACT-R is written in Common Lisp, and this makes it difficult to integrate with systems written in other programming languages. The options are to communicate through an interapplication communication protocol or to work entirely in Lisp. The latter route ensures more flexibility and is taken here, but this has also meant that the entire system incurs the slow performance of an interpreted language. Another difficulty lies

in the fact that cognitive modeling is not easy. It is, at best, an inherently iterative process and remains a difficult and intuitive, if principled, art. While the working model presented here succeeds in demonstrating the basic proposals of this report, substantially less than the full application task has been functionally modeled.

The task model tracing system is comprised of a prototyped application called a "mission planner," which implements an imaginary military planning task and an ACT-R model of a portion of this task. The ACT-R model implements the task-related cognitive skills the system is designed to simulate and is the part of the system referred to as the "task model" throughout this chapter. The working system also incorporates a copy of ACT-R's cognitive modeling environment, which is generally referred to as "ACT-R." The modeling environment is needed both to process the task model and for its command set, which is used to control the model's operations and to modify its state. The system is called a "task model tracing" system because task-related events are "traced" in the task model as they occur. The system's goal is to support a user interface that can accumulate and use task-related common ground with its user. As a domain, task-related common ground is taken to be knowledge of and about task-related interactions and the task itself. In general, the task model keeps track of the user's activities and the status of the task by monitoring application events. The user can also interact directly with the task model through an additional window, which augments the mission planner's user interface (Section 3.4.2). In this "task model interaction window," the user can prompt the task model to share its task-related information and make recommendations, which the user can then have the task model carry out. Actions carried out by the task model are also taken to be elements of task-related common ground. The task model interaction window and a number of additional programming constructs required to coordinate the operations of the task model and to facilitate code-level interactions between the application and ACT-R round out the task model tracing system's components.

Most of the effort required to implement the task model tracing system has been devoted to addressing the practical issues raised in deriving the task model. Typically, ACT-R models implement all or most of the declarative and procedural knowledge of interest and then use the architecture to model interactions among these elements. Ordinarily, knowledge that is implemented beforehand is derived through an iterative process of task analysis and model building, and this is the approach that has been taken here. Hence, the declarative and procedural representations that initially make up the task model tracing system's base of task-related knowledge more accurately represent the system's potential common ground with the user. The task analysis process must take into account a large number of considerations—the level of task-related detail, the interaction model, the user model, what the task model's computational functions are, and how these functions will be synchronized with the activities of the rest of the system. These and other related concerns are discussed in the sections that follow, beginning with a description of the mission planning task.

3.4.1 A Brief Description of the Application Task

The application's task domain is very roughly that of a military strike planning tool. Conceptually, the user is presented with a scenario in which a limited supply of tanks, fuel, and munitions must be used to engage one of three target destinations at various distances from a base of operations in an imaginary geography. All necessary information about mission resources and the destinations is available to the user in the planner's user interface. Many of the factors the user must take into consideration in planning a mission are interdependent, and to further complicate matters, tanks face a number of probabilistic risks of failure in both traveling to and engaging destinations. The application user interface, most of which is shown in Fig. 3, utilizes a standard point-and-click paradigm for user interactions and is composed of several dialog-box style windows in which the user can study and select destinations, equip and allocate tanks, and evaluate the success or failure of a mission. The user's most fundamental objective is simply to "go on a mission," and to do this, he or she must choose at least one destination and allocate at least one tank. Only a portion of the task represented by this objective has been thoroughly modeled in the work presented here—specifically that portion that entails the process of choosing a destination for the mission. Other facets of this objective have been partially modeled, but are not intended to be represented as complete.

3.4.2 Task Model Functions

The task model has several functions it must be able to carry out if the task model tracing system is to participate with the user on a cognitive level in the conduct of the mission planning task as a joint activity. In the language use

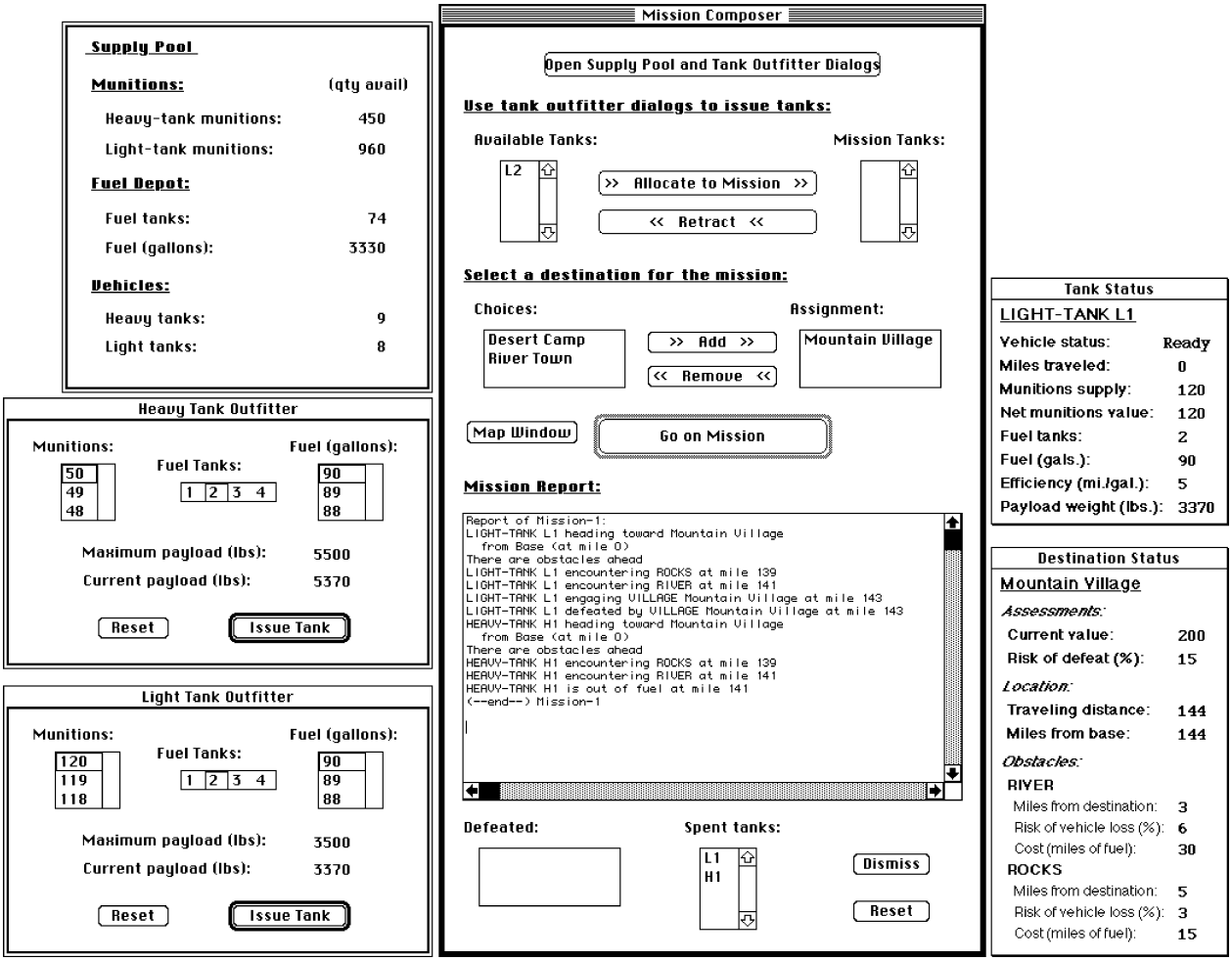

Fig. 3 — A screenshot of the prototyped mission planner application

approach to human-computer interaction emphasized in this report, interactions between the user and the computer are viewed as acts of communication. The user and the computer have strikingly dissimilar and limited ways of signaling each other in the task model tracing system's graphical user interface—the user's presentations are limited to the semantics assigned to the point-and-click format provided by the system, and the system's presentations are limited to what it can display and do—but an interaction model of this kind is nevertheless a language. In Clark's enlarged definition of language use, all forms of signaling must be included (Section 1.1).

In each of its interactions with the user then, ideally the task model tracing system is taken to be a participant in a joint action in which it is either the *addressee* or the *presenter** (i.e., signaler). In joint actions such as these between people, common ground is determined through subordinate joint actions the participants carry out in cognition (Sections 2.6.1 and 3.2). Thus, the task model's function is to simulate the cognitive portion of the system's part in each joint action that, depending on its role as addressee or presenter, respectively completes or prompts the user's cognitive actions.

For instance, when the system is the addressee, the user's input is taken to be the first part of a joint action the system is obliged to complete. As part of this obligation, ideally the system must do more than just receptively

*The term *presenter* is used here simply to denote the addressing party (i.e., the addressor). It is not specifically intended to connote the level of *presenting* used by Clark as a technical term to designate one of several subordinate actions a speaker (i.e., a presenter) carries out in a ladder of joint actions (Section 2.6.1, p. 13). *Presenter* is used here in favor of *speaker* or *signaler* because it best connotes both the demonstrative actions and display actions generally seen in human-computer interaction.

(and reflexively) process the user's input—a response that corresponds most closely in Clark's action-ladder view of joint actions (Section 2.6.1) to the subordinate, lower level joint actions of executing-and-attending and presenting-and-identifying, and is only a portion of what a joint action should entail. If the system-as-addressee is to fully complete the joint action the user has started, it must also carry out its part in the ladder's upper level, subordinate joint actions—computationally, the task model must *recognize* what the user is signaling and *consider* what the user is proposing. These are characteristically cognitive actions that people regularly expect their addressees to carry out in joint activities. Ideally, it should be no different in user interfaces. Similarly, when the system is the presenter, the explicit action it takes in the form of what it displays or does is taken to be the first part of a joint action that requires the user's participation to complete. Using the common ground it presumably has with the user, the system through its task model should simulate a cognitive component in this action that prompts the user's recognition and consideration of what (proposals) the system is signaling. Figure 4 illustrates the task model tracing system view of a human-computer joint action schematically.

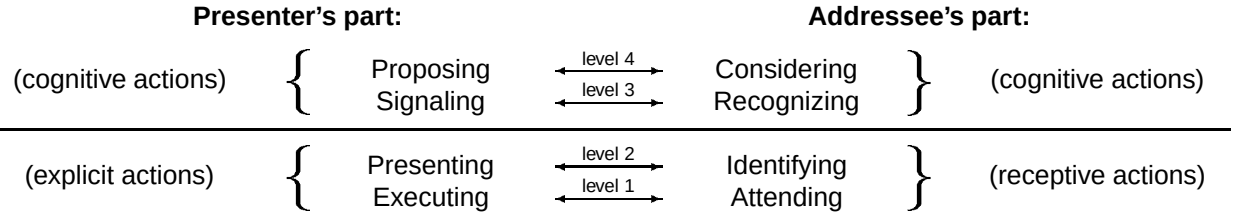

Fig. 4 — A schematic representation of a joint action with the task model tracing system

Before looking more closely at the task model's functions in joint actions when the system is the addressee and when it is the presenter, it should be noted that the task model tracing system's interaction model is essentially a *nonmixed initiative* design. By "nonmixed initiative," it is meant that all task-related activities must be initiated by the user, and that neither the application nor the task model are empowered to make independent decisions about the task and/or take actions on their own. In joint activities, the term *initiative* attempts to capture the notion of who initiates or proposes a joint action that takes the form of a *joint project* (Section 2.6.2) and whose goals are dominant (compare with Clark's dimension of *governance* in Section 2.3). Initiative is usually mixed in joint activities between people—at appropriate or even inappropriate times, any participant can take things in a new direction. In human-computer interaction, mixed initiative is seen whenever a system is designed to interrupt users (McFarlane, 1998) or otherwise act autonomously in interactive circumstances. (See Cohen et al., 1998 for a survey of issues in modeling initiative.) Since interruptions and other unprompted or unclear behaviors can be disorienting for users, it is generally recommended that users always be made the initiators of actions in interaction designs (Shneiderman, 1998). This recommendation has been followed for user interactions in the task model tracing system's interfaces—both the mission planner shown in Fig. 3 (p. 23) and the task model interaction window, which is described later in this section and shown in Fig. 5(b).

In the task model tracing work described here, the system is taken to be the addressee whenever the user initiates an interaction with the mission planner or with the task model directly in the task model interaction window. In its role as addressee, the task model takes each user interaction with the the mission planner and the system's response (essentially, application events and feedback) as a new element of common ground. So, for example, when the user clicks on a widget in the mission planner's Mission Composer window, such as the Map Window button in Fig. 3 that opens up a map of the task's imaginary geography (Fig. 5), the task model must note this and the result (the display of the map window), because these events are a *basis* for information that should now be common ground for both the user and the system. In addition, the task model should make note of any task-related implications of these events (i.e., *propositions*) that may reasonably be known or of use to the user. (Compare these functions of the task model with Clark's formal definition of common ground given in Section 2.5.1; also, see the material at the end of Section 3.3.1.) The most important implication of this sort that the task model keeps track of (or traces) in the task model tracing system is an assessment of the status of the task. In doing this, and in keeping track of interaction events in the mission planner when the system is the addressee, the task model, through its initial and established common ground with the user, maintains a record of the second and third parts of the joint activity's common ground at the moment—respectively, the *current state of the joint activity* and the *public events so far* (Section 2.3.2).

When the user initiates an interaction directly with the task model in the task model interaction window, he or she is considered to be *prompting* the task model to participate in the task as a presenter. The task-model-as-addressee in this case does not specifically note the user's prompt as a basis for common ground, but instead takes it as its cue to change roles and make a *presentation*. Ideally, task model presentations should be of use to the user and support the advancement of the task. With this in mind, but noting that each class of presentation behavior impacts on the complexity of the task model in terms of its information requirements and its organization, the task model tracing system implements its presentations in three forms—*reporting*, *advising*, and *doing*. The first two of these task model *presentation functions* are display-based and are always presented at the same time in the task model interaction window (Fig. 5(b)). Each is intended to give timely, task-related information to the user, as well as to reinforce each of the three parts of task-related common ground at the moment. Through its reporting function, the task model presents the user with a summary of task-related events that are relevant to the current status of the task; this corresponds to a selective representation of the *public events so far* as warranted by the *current status of the joint activity*. The system's *initial common ground* with the user—the first part of the task's common ground at any moment—is revealed in its reports through incidental references to elements of the user interface, specific interaction techniques, and features of the task, etc. Much the same is true for the presentations made by the task model's advising function. In this form of presentation, the task model attempts to contribute to the advancement of the task by recommending task-related actions the user may wish to take, based directly on the task model's assessment of the *current state of the joint activity*, and implicitly on its representation of the *public events so far*. Further, these recommendations are presented in an interactive format that allows the user, at his or her discretion, to have the task model carry them out individually.

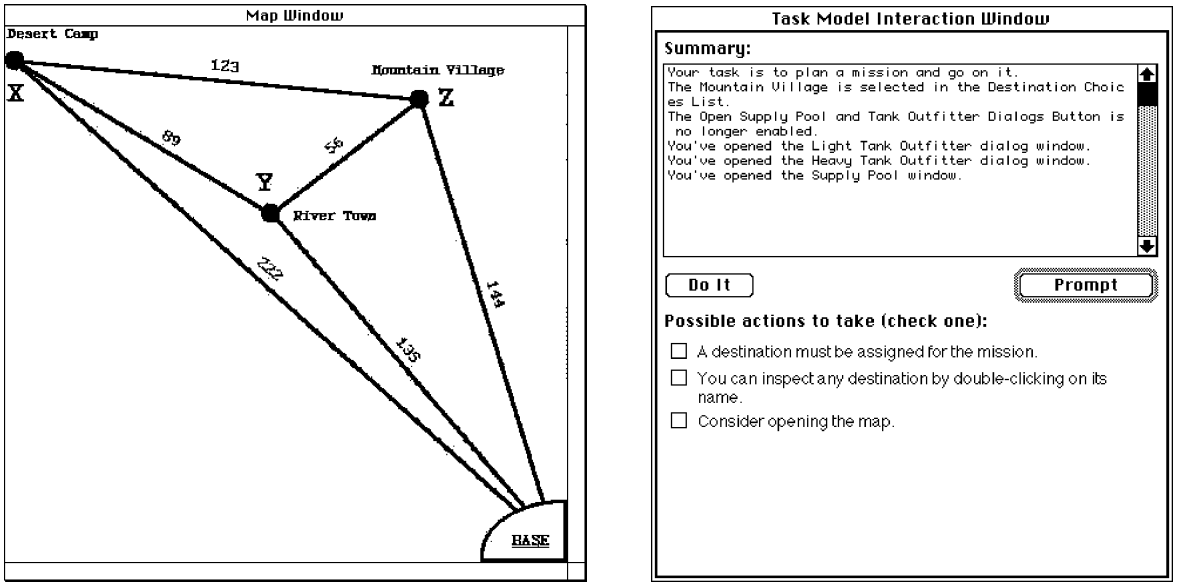

Fig. 5 — Screenshots of (a) the mission planner's map window and (b) the task model interaction window

The task model's ability to carry out actions it recommends constitutes its doing function. Carrying out task-related actions for the user is just as much a form of presentation as are user inputs because the actions themselves signal transformations in the task activity the participants can (and should) observe. Any such task-related event—not just the ones induced by the user—should serve as a shared basis for common ground between the user and the system. Consequently, as part of its doing function, the task model must be able to keep track of its own interactions with the mission planner and the resulting application events for essentially the same reasons that it keeps track of those due to the user—to maintain a current record of the system's task-related common ground with the user.

In summary, in its function as addressee, the task model simulates the cognitive component of the task model tracing system's joint actions with the user principally by keeping track of the importance of task-related application events and by assessing the status of the task. Computationally, these simulated cognitive actions are intended to

correspond to and complete those carried out by the user as part of the joint action that corresponds to each of his or her interactions with the mission planner. Conversely, in its function as presenter, the task model simulates the cognitive component of its presentations (as the first part of a joint action with the user) by using its representation of the task's common ground at the moment as a basis for summarizing the progress of the task and recommending appropriate courses of action. Further, the task model draws on elements in this same representation of common ground when it carries out any of its recommendations for the user. How the task model simulates these cognitive functions, both in terms of knowledge and process, and a number of related methodological issues are covered in detail in the next three sections, beginning first with a description of the methodology of model tracing and its use in the task model tracing system.

3.4.3 Model Tracing

It was mentioned early in Section 3.3 that a central claim of the ACT-R theory is that cognitive skills are realized by production rules (Anderson, 1993). There is a sizable body of empirical evidence to support this—as an early test of the psychological reality of production rules, ACT-R was used as the basis of a series of successful, computer-based, *intelligent tutoring systems* (Anderson, 1987). In these systems, production rules were taken to be the basic grain size of cognition, and the operational premise was that the skills being tutored ideally could be decomposed, cast, and monitored at this level. The tutors used a methodology called "model tracing" to follow the student's efforts and to compare them, production by production, to the idealized model of the skill being taught. Mistakes were recognized as erroneous productions that were similarly traced in a representation of the student, and this allowed the system to intervene with suggestions and explanations.

As interactive systems employing cognitive simulations, these tutors effectively had many goals in common with the work presented in this chapter. Not only were the tutoring systems in bona fide partnership with students, making the lessons legitimate joint activities, but much of the content of their interactions, and therefore their common ground, was cognitive in nature. Although the authors of these systems were in pursuit of a somewhat different agenda, the model tracing paradigm proved to be an elegant scheme for keeping the production-level computations of their lesson models congruent with student actions.

In the work described here, the model tracing paradigm is similarly used to monitor the user's and the system's task-related activities in the task model—although, not necessarily on a production-by-production basis and not necessarily for the same imperatives, hence the modified term *task model tracing*. However, there are many parallels. Just as a model tracing tutor employs an idealized representation of a skill and possible mistakes in its lesson model of the student, so the task model tracing system employs an idealized representation of the application task and the user's conceptual view in its task model. In both systems, model tracing is a reactive process—computations are carried out in response to user inputs—and in both, model tracing serves a dual purpose—to maintain an up-to-the-moment account of the status of the activity and to support the system's task-related interactions with the user.

Task model tracing and model tracing diverge mainly in terms of their agendas and their view of cognition. Unlike model tracing, task model tracing has no pedagogical agenda—the user is "always right" as a matter of policy (at least in this work) and the task model's only purpose is to assist the user in accomplishing the application task. Nor is task model tracing an attempt to verify a theoretical account of cognition. Interactions, and not productions, are taken to be the natural grain size of activities in application user interfaces; interactions inherently vary in scope and expressive power, and a trace of several productions may be required at the task model level to account for all that is implied by a given interaction. Task model tracing differs from model tracing in one further respect—what is traced. Only the student's cognition is of interest in model tracing—the computer's role in the tutoring process is viewed as part of the instructional framework but not of interest in what is being modeled. In contrast, the goal in task model tracing is to model the public and inferred status of the task as a joint activity being pursued by both the user *and the computer*. Consequently, the actions of both are monitored and traced in task model tracing.

3.4.4 Task Analysis and Task Modeling

Task analysis lies at the heart of model tracing schemes, and task model tracing is no exception. Generally, the term is taken to mean an analytical process of decomposing a procedural task and specifying its parts. Task analyses are

used by practitioners in many fields for a variety of purposes—to formalize a process, to explicate a poorly understood procedure, to determine the requirements of a design, and so on. In model tracing, and more generally in research-oriented cognitive performance modeling, task analysis is often used iteratively, first to make an educated guess at the requirements of a particular cognitive skill, and then to revise or refine the corresponding representation, realized in model form.

The goal of task analysis in task model tracing is to gather all of the information needed to create the *task model*—a cognitive model of an application's task, represented from the *perspective of the application*. The task model's purpose is to provide the application with a capacity to simulate task-related cognitive activities that may be of use to the user. To carry out this function, the task model must be able to work at a number of different levels, mapping task-related events onto a working knowledge of both the application's implementation of the task and the user's concerns, and supporting direct interactions with both the user and the application (Section 3.4.2). The scope of the task analysis process in task model tracing is consequently large and entails a number of conceptually different analyses (described as "phases" of analysis below). And, once the actual process of task modeling has begun, these analyses need to be revisited iteratively as the organization of the model takes shape and other practical considerations arise.

Task analysis for the purpose of cognitive modeling is essentially a process of decomposing information into declarative and procedural knowledge. If, for example, part of a task requires the user to press a button, a first pass representation of this would likely entail representing the button as a chunk, including perhaps its state of being pressed or not, and specifying an applicable set of production rules in simple "if-then" form. Since production rules represent when and how declarative knowledge is changed, the condition side of these rules would specify various declarative contexts, and their action side would show how the button press impacts other declarative representations, such as chunks corresponding to the immediate function and/or semantics of the button. From the user's point of view, there might also be other conceptual consequences of the button press, such as an understanding that a parameter for an indirectly related subtask has been changed, and this would be analyzed in much the same way.

For the application user interface to usefully simulate a set of task-related cognitive skills, the task model must first possess a core of background knowledge about the task and how it is implemented by the application. A basic representation of this knowledge can be developed from an analysis of the application itself, and this should be the object of the task analysis' first phase. In particular, the goal here is to characterize the task's procedural dimensions in terms of the implementation. For instance, if selecting a particular item is a procedural requirement of the task, the analysis must specify this in terms of how it is actually done. Building small, exploratory models of individual segments of the task with ACT-R is a useful adjunct to this phase of analysis.

To support a shared perspective with the user as it traces the rudiments of the task, the model must also be able to recognize and represent what is being done in terms the user can easily understand. Consequently, the next phase of the task analysis focuses on characterizing the task from the user's point of view. The job here is to develop an easily grasped conceptual representation of the task's components that can be unified with the task model's implementation knowledge. For instance, the model's task-level representation of the item selection event used as an example in the previous paragraph would likely be characterized in terms of the item's name, its role in parameterizing the next step in the task, and so forth. The user's characterization of this event, on the other hand, could be that it completes a milestone in the task (if in fact it does), and it is important for the task model to represent this. Goals, strategies, and meaningful sequences of interactions are all useful abstractions for users. Modeling the task in this way necessarily involves a degree of subjectivity, but the challenge is only to establish a viable degree of initial common ground with the user on these terms. Even a simple, carefully considered analysis can accomplish this.

The task implementation and user view models derived up to this point must next be unified and cast as a set of production rules and chunks to be processed by ACT-R. In anticipation of this effort, the task model's functional organization needs to be worked out. As addressee and presenter, the task model must be able to follow and reason about the task on an event-by-event basis, and contribute to the task's advancement when the task model tracing system is prompted for its input (Section 3.4.2). Although these are conceptually different functions, they are nevertheless interrelated. In particular, realizing and bounding the task model's functions in its role as presenter will largely depend upon the computations it carries out as addressee. More to the point, these functions, depending on their role, must be designed to work with either the mission planner or the task model interaction window. For instance, the task-related

knowledge derived up to this point will form the core of the task model's addressee function and must be able to respond directly to specific application events, for it is in this portion of the task model as a whole that the system's basic model tracing activities will occur. Hence, an organizational process must be imposed on the task model to coordinate the actions of its individual functions and to enable it to work as a whole with the rest of the system's application and user interface components. Simulating this as an additional cognitive process in ACT-R would go well beyond the goals of this work. The compromise employed here is to organize the task model into a partial hierarchy of goal-based *stages* that are invoked and managed directly by the application and the task model interaction window as appropriate. How this scheme works and is implemented is described in detail in Section 3.4.5. The model tracing core of the task model's addressee function is the first of these stages, and it's processing must be prompted by events in the mission planner. These factors must be taken into account as the knowledge derived so far is unified and rendered as an ACT-R model.

Additional procedural and declarative representations are needed to support and stage the participatory actions of the task model's presentation functions, and deriving these representations is the object of the remaining phases of task analysis. The task model's participatory actions—the presentation functions given in Section 3.4.2—require the model to be able to carry out three separate activities:

1. *Reporting*: assessing and summarizing the status of the task,
2. *Advising*: anticipating and recommending relevant actions the user may wish to consider taking, and
3. *Doing*: performing recommended actions at the user's discretion.

Each of these functions supports a different interaction goal and plays a role in establishing common ground with the user as well as advancing the joint activity of the task. All of the task model's presentation functions depend directly on its representation of common ground in the task, but are only needed when the user prompts the model directly for its input. Consequently, the task model stages its primary function—following the task—and its presentation functions—reporting, advising, and doing—separately. In particular, the model follows the task in response to application events and makes presentations in response to direct prompts. Internally, the task model's staging mechanisms are implemented declaratively as top-level goals that are operated on procedurally by stage-specific rule sets. Externally, the system's control of these mechanisms is supported with additional application and user interface code.

In addition to staging, though, another modeling strategy is needed to orchestrate the task model's presentations. At issue is how presentations are to be tied to the status of the task. In ACT-R, only production rules can carry out actions, so the statements displayed by the task model's reporting and advising functions must be produced by the application of procedural knowledge to the current state of the model's declarative memory (Section 3.3.1). In practice, this means that the model must contain a substantial number of production rules for making presentations. When the task model is prompted, it must determine which matters warrant summary and/or advice, and then fire the productions that match these matters and the task's current status. Later, when the status has changed and the model is prompted again, some of the same matters may warrant another presentation, but an entirely different set of productions will apply. In short, the model must contain many productions for each matter that warrants presentations—potentially, one for each change in the task's status! A more important consideration, though, lies in the problem of distinguishing one status from another. If the task is sufficiently represented, its status at any point will correspond to the state of the model's declarative memory. However, specifying each state in terms of the unique combination of task-related declarative representations that defines it is both unwieldy and impractical. A different approach is needed, and the one used here assigns names to individual task states and represents the global notion of the task's current status declaratively. Transitions are then managed by a corresponding set of production rules that are staged by the model as part of its addressee function after it has staged its primary subfunction of following the task. Figure 6 depicts the principal components in this process. From a modeling perspective, this strategy for representing task states and the status of the task makes it possible for an individual presentation rule that applies to a particular task state to be tied to it simply by specifying its name.

The third phase of task analysis is concerned with identifying task states and their corresponding transition rules. Before covering the process, though, recall that in the second phase of task analysis, the aim is to abstract important task-related concepts, such as goals and strategies and sequences of interactions, into representations that are likely to be familiar and easily grasped by the user. The rationale for making these representations a part of the task model's

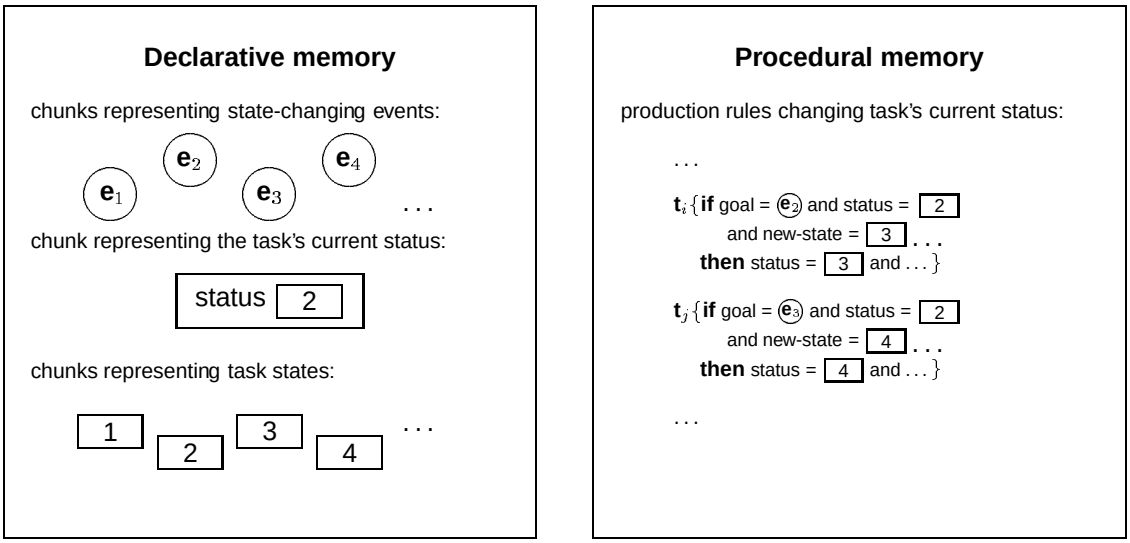

Fig. 6 — Representing task states and the task's current status in the task model. State-changing events and task states are represented as chunks in the task model's declarative memory. An additional chunk representing the current status of the task references the appropriate task state chunk in one of its slots. When a state-changing event occurs in the application, the chunk corresponding to it in the task model is made the goal. A specific set of production rules (designated $\mathbf{t}_i$ and $\mathbf{t}_j$ in this illustration) then carry out the appropriate transition. (The actual process is more elaborate than this simplified version indicates.) These actions are staged by the model as part of its addressee function. The task states shown in this illustration correspond to the notion of *task junctures* described later in this section on page 30 et seq.

reasoning process is common ground—the model is obligated to share the user's view if its participation in the task is to be merited (Section 2.5.1). However, it should be understood that none of these representations are ever placed directly before the user. Instead, these are the very matters with which the task model's presentation functions must work. Each serves in the task model proper as an anchor for all of the model's presentations on the matter it represents declaratively. In turn, each of these presentations is anchored to a task state. In the mission planner, for instance, choosing a destination for the mission is a fundamental part of the task, but only the notion of the destination needs to be represented declaratively in the task model proper to anchor the larger model's before and after presentations on the matter. Thus, evidence of the model's accumulated common ground with the user is exposed through its presentations. Figure 7 shows how the anchoring mechanism works. Each type of presentation function implemented in the task model tracing system addresses a different concern. The reporting function is intended to corroborate the user's understanding of the task, in part, with a summary of the public events so far that is relevant to the current status of the task, and the advisory function is intended to advance that understanding with appropriate recommendations. Hence, before the destination has been chosen, the model advises that choosing a destination is something that needs to be done; after the choice has been made, the model confirms the significance of the event in its summary of the status of the task.

Identifying task states, then, is essential for making relevant presentations. In the language use view, it is a fundamental part of representing common ground in the joint activity of a human-computer interaction task. If the representations developed in the first two phases of task analysis correspond most closely to the notion of *initial common ground*, modeling the task's status corresponds more closely to the notion of the *current state of the joint activity* (Section 2.3.2). In the application's user interface, both the user and the task model share an *external representation* of their joint activity (Section 2.3.2 and Table 5) that offers compelling evidence for their individual, but presumably mutual representations of the activity's—and hence the task's—status. Its *states*, as they occur, become a matter of public record—a part of what each participant presupposes (in theory) are the joint activity's *public events so far*. Consequently, an analysis aimed at identifying the task's states and their corresponding transition rules should begin with an analysis of how these states correspond to the states and transitions implemented in the underlying application. A practical concern faced in this next phase of task analysis, though, is to develop a strategy for defining task states that keeps their number in check and yet provides the task model with a viable basis for its participation in the joint

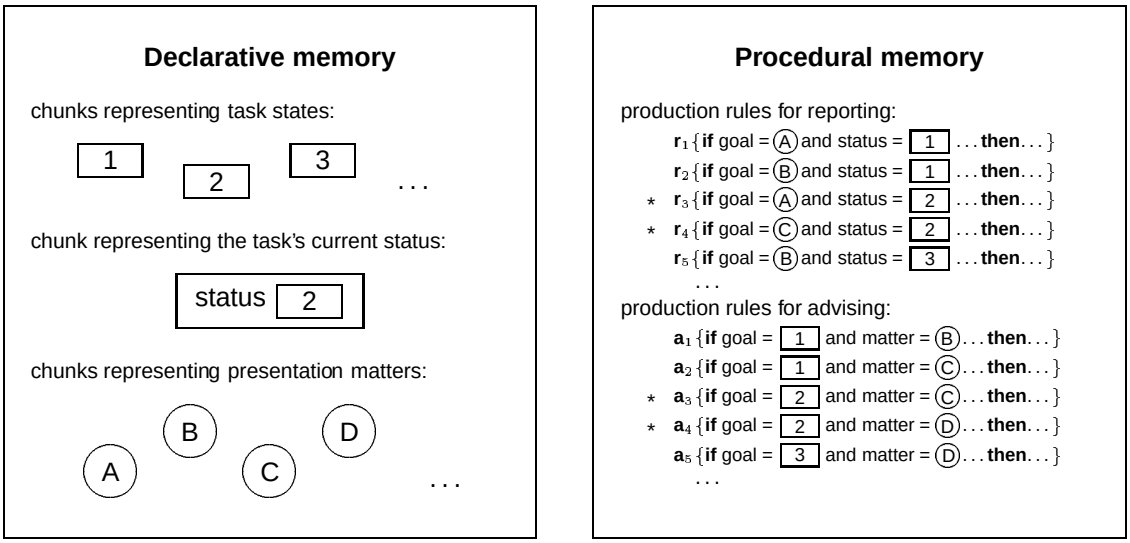

Fig. 7 — Anchoring production rules to presentation matters and the current status of the task. In addition to task states and the current status of the task (see Fig. 6, p. 29), matters that may warrant presentations—conceptual elements of the user's view of the task—are represented as chunks in the task model's declarative memory. The task model's display-based presentation functions (reporting and advising) are made up of many production rules (designated $\mathbf{r}_i$ and $\mathbf{a}_i$ in procedural memory in this illustration). Each rule is anchored the subject of its presentation and/or when it should be presented by specifying the corresponding presentation matter and task status chunks as a part of its condition for firing (i.e., in its **if** clause). Note that when a matter or the task's status is used as the task model's goal is a function of how the task model tracing system implements the presentation process (see material on the task model's presentation functions at p. 34 et seq. in Section 3.4.5). Depending on the goal (not shown in this illustration), the productions marked with an asterisk are those that are currently applicable to the state of the task model's declarative memory as it is depicted here.

activity. The approach adopted for task model tracing is to decompose the task into a set of goal oriented steps that are taken to be of interest to the user. It should also be recognized that interacting with the application is conceptually different from accomplishing the task—if, for instance, the application provides more than one way to accomplish any task-related step, there will not be a one-to-one correspondence between the two state spaces. Depending on the user interface, an analysis of this nature can quickly become a large project, and the challenge is to manage its complexity. It may not be possible or desirable to exhaustively determine all possible correspondences. In addition, the user may find some task states subjectively more significant than others—a point that is underscored by Clark when he describes how people keep track of activities by forming *annotated records* and *outlines* (Section 2.3.2).

The strategy developed here for identifying and modeling task states, introduces the term *task juncture* to denote any application state that is part of a direct path to the task's goal. An example of a juncture in the mission planner would be the application state reached by performing the actions required to choose the mission's destination. Junctures do not correspond to application states that are not on a direct goal path. Retrieving information about a destination may help the user to decide whether or not to choose it, but placing the application in this state does not move the task forward. An application's set of junctures are taken to be its task's conceptually significant states.

Task junctures are generally order dependent, but can also be order independent within groups. Figure 8 shows four examples of how application states can correspond to task junctures. All of the junctures shown in the first three examples (Fig. 8 (a, b, and c)) are order dependent, whereas some of those shown in the last example (Fig. 8 (d)), specifically junctures J_1 through J_7, are not. The ways in which application states combine, relative to their order independence, define the corresponding task junctures. In the simplest case (Fig. 8(a)), application states are linearly ordered and, so, correspond directly to order dependent task junctures: for linearly dependent junctures J_i and J_{i+1}, it is necessary to first reach J_i in order to then reach juncture J_{i+1}, and so on. The next two examples (Fig. 8(b and c)) show how order dependent junctures correspond to application states in other circumstances. In Fig. 8(b), application state s_j, which can be reached from s_i, is not task-related. In Fig. 8(c), application states s_i and s_j are functionally equivalent in terms of the task.

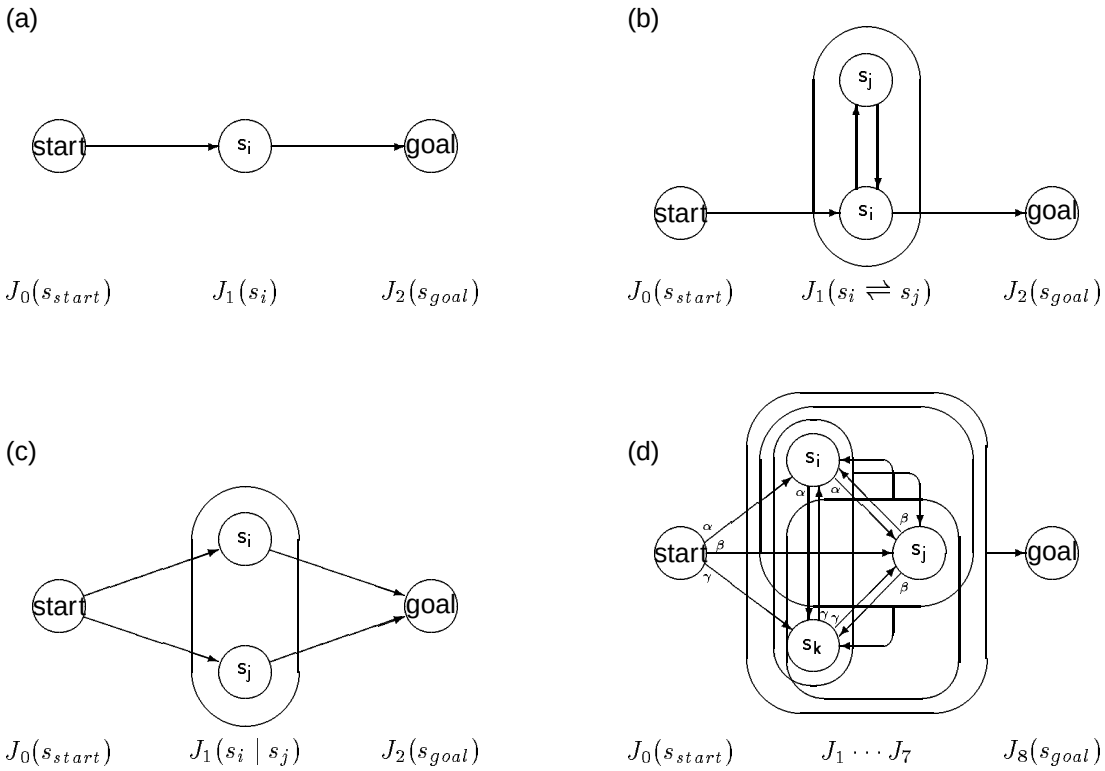

Fig. 8 — Four examples of how application states s may correspond to task junctures J (compare example (d) with Fig. 9)

Task junctures can also occur in groups corresponding to application states that are locally order independent. In general, for a set of n such states, there are $n^2 - 1$ corresponding task junctures. In the example shown in Fig. 8(d), the three states s_i, s_j, and s_k must all be reached before s_{goal}, the goal state, can be reached. Yet, each can be reached directly from s_{start} as well as from each other. Hence, these three application states are locally order independent. In this example, task juncture J_0 corresponds to s_{start}. But since s_i, s_j, and s_k are order independent, they correspond to a group of $3^2 - 1$ junctures that are determined as follows. Junctures J_1 J_2, and J_3 correspond to having only reached the respective, single states of s_i, s_j, and s_k directly from s_{start}. Since, from each of these junctures, two more application states must still be reached—in any order—before s_{goal} can be reached, the next three junctures, J_4, J_5, and J_6, respectively correspond to having reached the first of these two remaining states. Put another way, task junctures J_4, J_5, and J_6 correspond to having reached, respectively, the two states s_i and s_j, s_j and s_k, and s_i and s_k, regardless of order. (For instance, J_4 corresponds to having reached state s_i first and then s_j, or having reached s_j first and then s_i; either way, state s_k must be still reached next before s_{goal} can be achieved.) Finally, then, juncture J_7 corresponds to having reached all three of s_i, s_j, and s_k, again, regardless of order. Note that junctures J_1, J_2, and J_3 as a group are order independent; so are J_4 and J_5 as a group when approached from J_1; and so on. The transitions in Fig. 8(d) are shown more explicitly in Fig. 9.

The conditions that determine when each task juncture has been reached must also be identified and cast as production rules. With these rules in place, the task model is able to represent the situation at any point in the user's performance of the task in the form of a juncture. Managing complexity in this phase of analysis, though, is essentially a process of compromise. A particularly useful approach can be to identify junctures hierarchically, identifying first the major components of the task and then working through specific application states at progressively lower levels. It is also important for the task model to represent application states that are not task junctures (such as s_i in Fig. 8(b)), especially when they are germane to the user's understanding of the demands of the task. One strategy for modeling such states is to treat them as conceptual elements of the task states to which they are connected; an example of this will be discussed in Section 3.5.

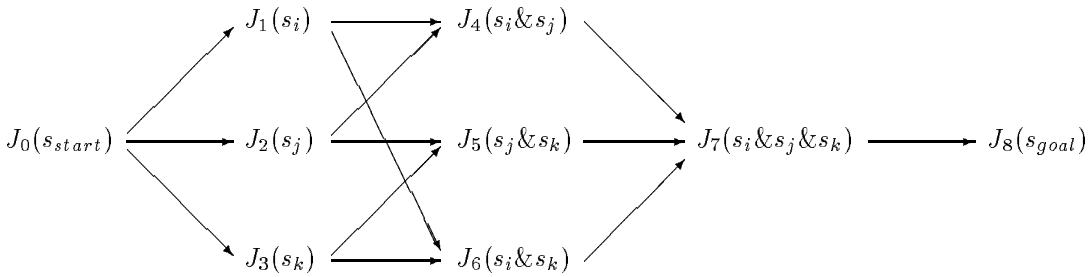

Fig. 9 — A different view of Fig. 8(d)

The object of the fourth—and last—phase of task analysis is to develop the task model's substantive presentations. In the present work, the model's reporting function is composed of production rules whose actions include direct output of summary text. The advisory function has been organized differently—it is composed of production rules that can be fired. Once these components of the larger task model have been developed, they will constitute the task model's actual presentations. In practice, this phase is the most iterative and interwoven analysis of the four because it depends heavily on the emerging organization of the task model and a variety of practical considerations inherent in the application environment. Examples of how these presentations appear and are used in the mission planner are given in the next section.

3.4.5 A Description of the Full System in Operation

Conceptually, the task model tracing system described in this chapter is composed of:

1. *A host application*: the prototyped mission planner.
2. *ACT-R* : the ACT-R simulation of the human cognitive architecture.
3. *A task model*: the declarative and procedural representations of the application task, derived iteratively through task analysis and model building, which are processed by ACT-R .
4. "*Glue*": supplemental user interface and system code and routines necessary to permit the user to interact with the task model and to permit the task model to participate in the task.

Except for the task model, which is written in the ACT-R cognitive modeling language, each of these components is written in Common Lisp, and all function together in the Lisp environment as a single, unified process.

The host application's user interface is made up of several windows, all but one of which are shown in Fig. 3 (p. 23). The remaining application window, which shows a map of the mission's geography, is shown in Fig. 5(a). The other window shown in this figure (Fig. 5(b)) is the "task model interaction window." This window, which permits the user to interact with the task model, is not part of the application proper but augments its user interface and is available to the user at all times. When the system is launched, only the large "Mission Composer" window and the task model interaction window are opened. The application's other windows are opened during the course of the task through button presses or by double clicking on individual items in the Mission Composer's tank and destination lists.

3.4.5.1 Basic Model Tracing Activities

As the user interacts with the application and application processing occurs, the application also drives the task model tracing process. It does this each time a task-related event occurs with a glue routine called *update-model* that runs ACT-R (and hence, the task model). Ordinarily, ACT-R is run until the model has completed the immediate process of updating its representation of the task. This activity of tracing application events in the task model is repeated steadily until the user chooses to interact with the task model through the task model interaction window (described below).

To stage both its model tracing function and its other activities, the task model uses a special hierarchy of goals. The topmost goal in this hierarchy is a chunk of type *hold* that carries the current task juncture with it in one of its

slots. This is the same chunk shown in Figs. 6 and 7—labeled "status" for clarity—that is used by the task model to represent the current status of the task. The *hold* chunk is privileged because it is never removed from the goal stack. It is used instead to indicate when running the model should be stopped. Thus, whenever the task model is at rest in the context of the full task model tracing system, ACT-R's goal stack is empty except for the *hold* chunk, which is deliberately left on top. To run the model, the *update-model* routine relies on a subordinate glue routine called *stop-at-hold*. This routine repeatedly cycles the model using ACT-R's *run* command whenever the *hold* chunk is *not* on top of the stack. To ensure that the model will run, the *update-model* routine must always push another goal onto the stack before it invokes *stop-at-hold*. The goal it installs is a chunk of type *bookkeeping*. This and two other chunks, one of type *assessment* and the other of type *report*, make up the next level in the model's stage-related goal hierarchy; this hierarchy is shown in Fig. 10. An instance of each of these chunk types is used to initiate a different stage of the task model's activities—the *bookkeeping* and *assessment* stages carry out the task model's responsibilities in its role as addressee, and the *report* stage initiates the task model's presentation functions.

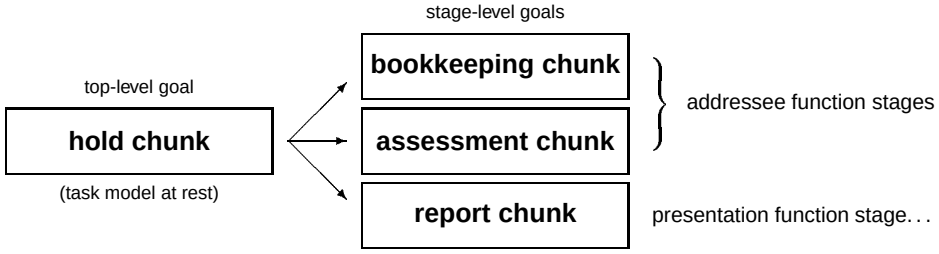

Fig. 10 — Task model stages and their relation to the top-level goal and the task model's addressee and presentation functions

In order for the task model's representation to remain fully abreast of task-related developments in the application, it is necessary for the model to have essentially unrestricted access to the application itself. This issue was raised at the beginning of Section 3.4. If the host application and ACT-R were to operate in different language environments, it would be necessary to *factor* the entire application so that it could be fully accessed and controlled by the task model via an interprocess communication protocol (see, for instance, Ritter and Major, 1995). Since this approach also would have required the host application to be implemented from the ground up, it was not pursued in the present work in order to focus more directly on the issue of modeling cognitive aspects of common ground in a human-computer interaction task.*

Although the task model is effectively *embedded* in the application, it is nevertheless a separate process. It gains access to the application by maintaining copies of task-related application objects in specific slots set aside for this purpose in its declarative representations of these conceptual entities. When a task-related application event occurs, it is incumbent on the application to both run the task model and to present it with copies of any application objects involved in the event. This is accomplished one object at a time through the *update-model* routine and a chunk of type *code-reference* that is used by the bookkeeping stage solely for this purpose. In addition, each time the application invokes the *update-model* routine, it must also identify the chunk in the task model that corresponds to the object it is presenting by specifying its *label*—a slot value that is associated with all such conceptual representations in the model. With this information, the task model tracing process can begin. The *update-model* routine does two things with the information it has received from the application—it assigns the copy of the application object it has received to the task model's *code-reference* chunk, and it locates the chunk whose label it has been given and modifies it to be *attended to* by the model in the upcoming round of model tracing. It then uses a subordinate routine called *push-chunk* to push the *bookkeeping* chunk onto ACT-R's stack and invoke the *stop-at-hold* routine to run the model.

Two stages are involved in the basic model tracing activities the task model carries out in its role as addressee—the "bookkeeping" stage and the "assessment" stage. The latter is devoted primarily to handling transitions between task junctures. In the bookkeeping stage, the chunk specified by the application that is to be attended to is immediately identified and bound to a slot in the *bookkeeping* chunk. This same chunk (not the *bookkeeping* chunk) is then made the *model tracing goal* by pushing it onto the stack. If it has not already been done in an earlier round of bookkeeping,

* An interprocess communication approach was, in fact, initially taken in my earliest task model tracing work. Few if any fully factored systems were found to exist.

the application object carried by the *code-reference* chunk is also bound to a corresponding slot in the model tracing goal. The core model tracing production rules then take over. During this phase, the model typically pushes and pops an additional goal or two as needed to fully carry out its tracing activities. The production rules involved in this process use the model tracing goal's copy of its corresponding application object and possibly those of other chunks to verify the immediate state of the application. Junction transition events are also identified in this stage. The application represents these events explicitly by creating declarative representations for them and passing each event's label to the *update-model* routine. A corresponding set of bookkeeping stage production rules intercepts these chunks when one of them has been made the model tracing goal and primes the model declaratively to change junctures in the assessment stage that follows. (Figure 6 shows the state-changing process.)

As the bookkeeping stage completes its model tracing activities, any task-related goals that have been pushed onto the stack are removed, including the model tracing goal. This restores the *bookkeeping* goal to the top of the stack. The model tracing goal will be needed again during the assessment stage, so it is now bound to the assessment stage's goal chunk (the *assessment* chunk). The bookkeeping stage completes its activities by "cleaning up" the *bookkeeping* and *code-reference* chunks (binding their critical slots to neutral values, which readies them for the next round of bookkeeping) and then replacing the *bookkeeping* goal chunk on the stack with the newly configured *assessment* goal, signaling the beginning of the assessment stage.

The task model now turns its attention to assessing its representation of the task's status and to managing its support for the presentation functions. The bookkeeping stage's model tracing goal is pushed back onto the stack and, at the same time, a copy of this chunk is added to an external list called the **report-list** that is used by one of the presentation functions' glue routines in the "report" stage (see below). The reinstated model tracing goal is then evaluated by a specific set of assessment stage production rules. All but one of the rules in this set are anchored to the current task juncture, and all are part of a larger set of rules that represent transitions between task junctures in the task model. (Essentially, the production rules denoted $\mathbf{t}_i$ and $\mathbf{t}_j$ in Fig. 6 represent members of this set.) When the goal represents a transition event, the task juncture indicated by the corresponding transition rule is installed in the privileged *hold* chunk—the task model's topmost goal—by a further set of assessment stage rules dedicated to this purpose. If no transition event has occurred, a fall-back rule—the one not anchored to the current juncture—is applied. In either case, the model tracing goal is then popped, and the assessment stage completes its work by cleaning up its goal chunk and any other utilitarian chunks it has used in the process. The *assessment* chunk is then popped, restoring the *hold* chunk to the top of the stack.

3.4.5.2 Presentation Functions—Reporting, Advising, and Doing

Up to this point, only the task model's passive activities have been described. To interact with the task model, the user presses a button labeled "Prompt" in the task model interaction window (Fig. 5(b)). This button invokes a glue routine called *prompt-task-model* that is one of several used to support the task model's presentation functions. Prompting the model results in a two-part display of information in the task model interaction window. Both the model's reporting and advisory functions are invoked. In the upper half of the window, the model's report function presents a summary of the task's status; in the lower half, the model's advisory function presents an interactive list of recommended task-related actions.

Although the task model carries out its model tracing activities—its addressee function—essentially as a running model, the major part of its presentation functions is carried out in a more fragmented manner as an iterated series of short model runs coordinated by glue routines. The *prompt-task-model* routine begins the presentation process by calling a subordinate routine named *report-task-status*. This routine initiates the model's "report" stage by pushing the last of the model's three stage-related goals—the *report* chunk (Fig. 10)—on the stack and running the model with the *stop-at-hold* routine. A dedicated production rule immediately flags this goal to indicate that the report stage is in progress and then replaces it on the stack with the chunk that corresponds to the current task juncture. When *stop-at-hold* commences the next ACT-R cycle, a production rule in the report stage's procedural memory matches the juncture and issues a corresponding sentence-length statement that broadly summarizes the task's status and then pops the goal, exposing the *hold* chunk. The summary sentence is not immediately displayed, but is stored in a buffer called **report-buffer** which is displayed all at once at the end of the report stage.

With the *hold* chunk now on top of the stack again, *stop-at-hold* ceases to run the model, and control is returned to the *report-task-status* routine. A subordinate routine named *get-reports* is used next to generate any additional summary statements that may be relevant to reporting the current status of the task. The *get-reports* routine does this using the **report-list** of model tracing goal chunks that is managed by the assessment stage (see above). These chunks are the presentation *matters* depicted in Fig. 7. One at a time, each chunk in the list is pushed onto the goal stack, and the model is run via the *stop-at-hold* routine. This allows the report stage's procedural memory to match the goal and issue a summary statement, just as was done at the beginning of the report stage with the current task juncture. If for instance in Fig. 7, the matters named **A** and **C** have both been added to the **report-list** during the assessment stage, and the current status of the task is **2**, as is shown, then the production rules $\mathbf{r}_3$ and $\mathbf{r}_4$ marked with asterisks in procedural memory would apply in succession during this process. In general, the report stage's procedural memory is composed of rules that handle all of the model's juncture dependent and juncture independent reporting. Each goal is immediately popped after it has been matched. Any corresponding contributions to the status report are also buffered in the **report-buffer** list.

When the *get-reports* routine completes its work, control returns again to the *report-task-status* routine. The *hold* chunk is again on top of the stack, and now *report-task-status* initiates the end of the report stage by pushing the *report* chunk back onto the stack and running the model. Another dedicated production rule unflags the *report* goal, indicating that the report stage is now complete, and then removes it from the stack. With the *hold* chunk exposed again, control now returns to the *prompt-task-model* routine, which immediately displays the contents of the **report-buffer** in the upper portion of the task model interaction window, completing the actions of task model's report function (Fig. 5(b)).

The *prompt-task-model* routine turns its attention next to orchestrating the task model's advisory function. It does this by calling a subordinate routine named *anticipate*. In the task model described here, the advisory function is conceived as a subset of the model's production rules that match the *hold* chunk when it is the goal and the model is at rest. Since the *hold* chunk carries the current task juncture with it in one of its slots, an advisory function rule that corresponds to a particular task state only needs to specify the corresponding juncture in its goal condition. The *anticipate* routine takes advantage of this design by forcing ACT-R to generate its conflict set (Section 3.3.2) without running the model. Again, for instance, if in Fig. 7 the current status of the task is **2**, as shown, the production rules $\mathbf{a}_3$ and $\mathbf{a}_4$ marked with asterisks in procedural memory would make up this conflict set. The advisory statements that the model presents for a given state of the task are carried by the corresponding productions in a technical feature known as their "documentation string." Once the conflict set has been generated, the *anticipate* routine extracts each production's documentation string. These are then collectively passed to a subordinate routine called *print-anticipation-explanations* that handles their display as a group of checkboxes in the lower half of task model interaction window (Fig. 5(b)).

The task model's advisories are presented as a suggestion list of recommendations that the model anticipates will be informative and will contribute to the advancement of the task. Each recommendation also represents an action the model can carry out for the user at his or her request. Collectively, these actions, and the procedural and declarative representations in the task model that are needed to carry them out, constitute the third of the task model's presentation functions—its "doing" function (see Sections 3.4.2 and 3.4.4). The *print-anticipation-explanations* routine configures the advisory list, as a practical matter, so that only one recommendation can be checked at a time. When and if the user wishes to have the task model carry out a recommended action, he or she clicks on its corresponding checkbox and then presses the task model interaction window's "Do It" button (Fig. 5(b)), which invokes a glue routine called *force-rule-to-fire*.

Since the *hold* chunk is still on top of the stack, after the *force-rule-to-fire* routine has identified the production rule that is to be fired, it sidesteps the *stop-at-hold* routine, makes the rule known to ACT-R, and runs ACT-R directly for one cycle. This fires the rule. Each of the production rules that make up the task model's advisory function, when fired, immediately pushes a new goal onto the stack that identifies the specific action the model is to carry out next. After this occurs, the productions that make up the task model's doing function take over. With the *hold* chunk no longer on top of the stack, *force-rule-to-fire* now uses the *stop-at-hold* routine to run the model. The action represented by the goal is then carried out by a specific group of rules that must address the parameters of the action and drive the application directly. The parameters of the action must be derived from the task's accumulated common ground, and an example of how ACT-R can be used to accomplish this is described below in Section 3.5. Driving the application

directly poses a different problem, since the application itself ordinarily drives the task model's model tracing function through its invocations of the *update-model* routine. The solution is to suspend the *update-model* routine's ability to call *stop-at-hold* while the *force-rule-to-fire* routine is in effect, and to make it the doing function's responsibility to carry out the model tracing activities that would otherwise be invoked by the application. This allows the task model to trace its own task-related actions.

3.4.5.3 Summary of the System's Operation

Figure 11 presents a conceptual overview of the task model tracing system, showing how its major components—the mission planner application, ACT-R, the task model, and supplemental system code and routines, referred to as "glue"—are related to each other and interact.

Through its task model, the task model tracing system implements an interactive cognitive simulation of a portion of the mission planning task described in Section 3.4.1. The simulation is an ACT-R process that models the task as a joint activity in which the system participates in joint actions with the user as either addressee or presenter, depending on the nature of the interaction. As the addressee, the task model passively accumulates task-related common ground

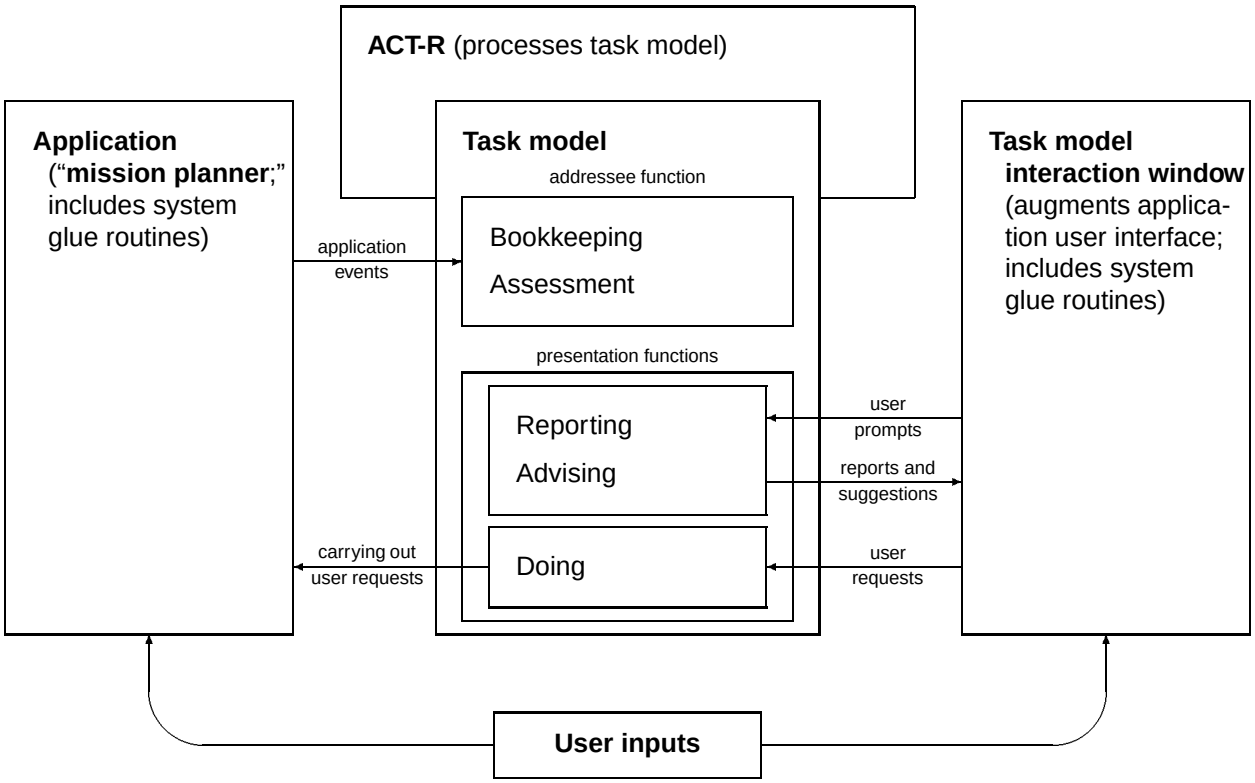

Fig. 11 — A high-level view of the principal components of the task model tracing system. Through its task model, the task model tracing system implements a cognitive simulation of the process of accumulating common ground with the user in the mission planner task as an ACT-R process. Using a number of supplemental "glue" routines, the system responds to user inputs in both the mission planner's application user interface and the task model interaction window, which augments it. As addressee, the task model responds to application events by staging its bookkeeping and assessment functions, which carry out the system's core task model tracing activities. When the user prompts the task model directly in the task model interaction window, the task model's reporting function is staged and then its advising function is carried out. The resulting presentations—a summary of the task and a suggestion list of recommended actions the task model can also carry out individually for the user at his or her request—are shown in the task model interaction window. When the user chooses to have the task model carry out a recommendation, it's doing function fires the production rule associated with the recommendation, and then carries out the request by interacting directly with the application. Interactions between the task model and the application are also traced by the task model's addressee functions (compare with Fig. 1).

with the user in response to his or her inputs, which are modeled as presentations. As the presenter, the system's initial and accumulated common ground is then used to determine what information should be useful to display and how the system should proceed if the user chooses to have the system carry out a recommendation it has made.

The task model's addressee function is driven by the mission planner when task-related application events occur in response to user inputs. To do this, the application is modified to invoke and supply model-specific, task-related information to a system glue routine called *update-model* that serves as a control interface between the application and the task model. When the task model is run by the *update-model* routine, its addressee function is orchestrated with a hierarchy of special goals that stage the system's basic model tracing activities. The topmost goal in this hierarchy, the *hold* chunk, is used to halt the running model and is never removed from ACT-R's goal stack. This same chunk is also used by the model to specify the current status of the task as a task juncture (Section 3.4.4). Two of the three goals at the next level in the hierarchy are used to carry out the addressee function. (The third goal, described below, is used to stage a presentation function.) During the "bookkeeping" and "assessment" activities staged by these goals, the task model follows the task and accumulates task-related common ground by tracing application events and updating its representation of the task and the task's status.

In addition to working with the mission planner, the user can interact directly with the task model at any time by prompting it in the task model interaction window, which augments the application's regular user interface (Fig. 5(b)). When prompted, this window displays a summary report of task activity and an advisory list of task-related suggestions. The suggestions are recommendations about how to proceed in the task that are presented interactively as a group of mutually exclusive checkboxes. Individual recommendations can be carried out by the model at the user's discretion by choosing one in the task model interaction window and pressing the window's "Do It" button.

The summary reports and recommendations the task model displays in the task model interaction window and the set of actions it can carry out for the user collectively make up the task model's presentation functions. Each of these functions is carried out by the system in a different way. The report function is staged by the task model using the third of the three goals under the *hold* chunk in the hierarchy of goals used in staging the model's bookkeeping and assessment activities, but is controlled by a glue routine called *prompt-task-model* that runs when the task model interaction window is prompted. The task is then summarized by pushing a series of goal chunks accumulated in the assessment stage onto the stack and running the task model incrementally; report function production rules that are applicable to these goals generate each segment of the report. The task model's advisory function is also coordinated by the *prompt-task-model* routine. Presentations of recommended, task-related actions are modeled as a set of production rules that match the *hold* chunk—which specifies the task's status—when the task model is at rest. Individual rules relevant to a specific task state are anchored to that state by specifying the appropriate task juncture in their goal condition. These rules are then identified in the *prompt-task-model* routine, and their corresponding presentations are displayed as a set of checkboxes, by forcing ACT-R to passively generate its conflict set—the set of production rules that are applicable to the current goal. The task model implements its doing function—its capacity to individually carry out these recommendations—with the help of an additional glue routine, *force-rule-to-fire*, that forces ACT-R to fire the associated production rule when the user checks a recommended action and presses the Do it button. This pushes a goal representing the action the task model is to perform onto the stack, and the running model then carries out the action directly by interacting with the application as a user. Since all task-related application events are taken to be a basis for common ground, the task model must also trace its own presentation-oriented interactions with the mission planner as part of its doing function—just as the user presumably monitors his or her own user inputs. Section 3.5 presents two different examples of how the task model is able to use common ground in its presentations.

3.5 Common Ground in Task Model Presentations—Two Examples

The central premise behind the idea of task model tracing is that human-computer interaction can be usefully likened to a joint activity between two people. Although in one sense an interactive application is only a tool, in another sense its computational nature makes it a legitimate—albeit limited—participant in the activity it has been designed to support. What limits an application more than anything else in this role is a general lack of the kinds of cognitive skills a person in its place could be expected to possess. Although task model tracing is certainly an attempt to examine the nature of this limitation through cognitive modeling, the larger concern in the work presented here is not so much with *how* such skills can be simulated, as with *what* skills are needed in the first place. Clark's work emphasizes that what participants in joint activities make use of are cognitive skills that support the possession,

accumulation, and use of common ground—and that it is through the exercise of these skills that advancement in joint activities may be characterized. As part of their common ground, people count on each other to possess such skills and look for evidence of their presence in each other's presentations in order to capitalize on them. When these skills are found to be wanting or missing—in effect, when there are imbalances in common ground—communication is hampered and joint endeavors become problematic. The more capable participant must work harder and be more resourceful to bring about the goals of the joint activity—and this is exactly the situation a user often faces in the context of human-computer interaction.

If cognition is to be simulated in a user interface, then it must be designed to do more than simply reason independently about the task—to be credible and useful, it must be designed to convey the extent of its common ground with the user through its presentations. This works at several levels. For instance, it can be assumed that the user is interested in how the application can assist in performing the task, so the application should be prepared to make this sort of information readily available. As the task advances, the user's representation of it as a joint activity steadily accumulates. It should be easy for the user to corroborate this understanding with the application's own version of things—what has been done and by whom. And in instances where an action the user might ordinarily perform has been delegated to the application, there may be parameters whose salience can be determined on the basis of the accumulated common ground, and the application should choose accordingly. Two examples detailing how the task model for the mission planner behaves in specific situations, one computationally straightforward and the other exploiting one of ACT-R's subsymbolic processing mechanisms, are given here to illustrate how the task model tracing work described in this report seeks to address these points.

The first example concerns the mission planner's map window (Fig. 5(a)). This window is not displayed when the application is first opened, but access to it is available by pressing the Map Window button in the mission planner's large Mission Composer window (Fig. 3). When the user prompts the task model directly and the map window has not been opened yet, one of the suggestions the task model's advisory function presents is that the user should consider opening it. Opening the map is not an action that directly moves the task forward, but it is nevertheless an important part of the application's support for the task because it presents the user with a useful representation of the mission geography. Indeed, once the map window has been opened, a task analysis should document that there are a number of task-related propositions about mission destinations that can now be inferred by the system to be common ground on the shared basis of what is shown in the map, such as, for example, each destination's relative distance and compass direction from the "Base" shown in the lower right corner. Recall that Clark formally defines common ground in terms of propositions that follow from shared bases (Section 2.5.1). Although this particular information is not currently a part of the task model, other elements of common ground related to opening the map window are. The act itself of opening the map window is a public event, with certain characteristics, that readily serves as a shared basis for propositions that both the user and the system should hold in common after its occurrence. Since the task model can also open the window when its recommendation to do so is shown in the task model interaction window, one characteristic of the event after it has taken place is simply who carried it out—the system or the user. Another is the simple fact that the window is now open. As a participant in the joint activity of the task, how the task model uses these and other mundane but task-related details in any of its subsequent presentations is important because of the evidence of common ground this provides for the user. Thus, once the map window has been opened, unless it is closed again, the system appropriately no longer includes a recommendation to open it in its advisory list. Instead, to corroborate the user's own record of what has transpired—the public events so far—a reference to the window's having been opened, and by whom, is presented in the task model's summary report. If the user has subsequently closed the map, the summary report reflects this too, but the task model also adds a subsequent recommendation about looking at the map again to its advisory list.* What is important about this example is that it demonstrates how task-related common ground can be modeled and used by a user interface. As a shared basis, each characteristic associated with a publicly occurring application event—in this case, opening and closing the map window—can be identified through a task analysis. Through further analysis, how these characteristics can be used by both the user and the system to advance the task can also be identified. A computational model can then be developed for the system's use of these

*When the user has opened the window, the task model's summary reports, "You have opened the map window." When the action has been carried out by the task model, the report reads, "The system has opened the map window for you." After the map has been closed, the task model reports, "The map window has been opened and closed," and its advisory list recommends, "Consider looking at the map again." (Note that this is different from the initial recommendation, "Consider opening the map.")

elements of common ground in order for it to participate in the joint activity of the task. When interactions between the user and the system are interpreted as joint actions, it is computations such as these, which functionally simulate cognition when the system is the addressee and when it is the presenter, that serve to complement (and complete) the cognitive skills the user naturally brings to the task (compare with Fig. 4).

The next example looks at how one of ACT-R's subsymbolic processing mechanisms can be used to compute a form of salience on the basis of accumulated common ground. The destination assignment is one of the basic decisions that must be made in carrying out the mission planning task. Before this decision is made, it is possible that the user will first want to learn something about each of the destinations. In addition to what can be gleaned from the map, destination status information is available in the application's "Destination Status" window when the user double-clicks on any of the names shown in the list labeled "Choices:" in the Mission Composer window (Fig. 3).* If the user prompts the task model before he or she has chosen a destination by moving it to the list labeled "Assignment:" in the Mission Composer, the model will recommend in its advisory list that a destination must be assigned for the mission. In addition, if the Destination Status window has not been opened yet, the model will advise that this can also be done (these advisories can be seen in Fig. 5(b)). Should the user choose to pursue either of these recommendations through the task model interaction window, it will be necessary for the model to make a choice for the user. In other words, both actions have a destination parameter, but which destination should the model choose? The most interesting solution to this problem from the point of view of the task's common ground is to choose the destination that is the most salient on the shared basis of its interaction history—here, when and how many times each destination has been explicitly part of a user interaction. If there is no such history, then the choice is arbitrary. But if the user has clicked on any of the destination names in the Mission Composer, and/or has already investigated any of them in the Destination Status window, these events can be taken to be in the joint activity's common ground, and their occurrence can be modeled as a basis, albeit imperfect, for inferring the user's choice. Thus, to keep track of the cumulative significance of these events, ACT-R's subsymbolic chunk activation mechanism (Section 3.3.2) is used by the task model as a measure of the user's interest in each of the destinations so far in the task.† As currently modeled, each destination begins in the task with an equal amount of activation, and these values then vary in time as the model runs according to their frequency and recency of use. In the eventuality that the user first examines some or all of the destinations, and then decides to have the system carry out the assignment, the production rule in the task model's doing function responsible for parameterizing the action capitalizes on this component of declarative learning in ACT-R's theory of memory retrieval and matches the most active destination as one of its conditions. ‡ In carrying out this part of the task after the user has presented evidence of his or her disposition in the matter, the system in effect faces a coordination problem—which destination does the user *expect* to be assigned?—that should be solvable on the basis of common ground. What the system and the user require is a coordination device that is jointly salient with respect to their current common ground, and for the task model, given the limitations of the interaction medium, the most active destination readily meets this criterion (see *principle of joint salience* in Section 2.4.1).

The two examples presented here are intended to demonstrate both the viability and efficacy of a language use approach to human-computer interaction. In the preceding exposition of task model tracing, elements of Clark's theory of language use have served throughout as the motivating basis for the design and function of an application task model written in ACT-R that is used as a means for simulating a set of task-related cognitive skills in the application's user interface. In particular, from a design perspective, interactions between the user and the system can be usefully viewed as a layer of action in which the user and the system are participants in a joint activity whose principal goal is the conduct of the application's task. Interactions between the user and the system in the course of this joint activity can be interpreted as joint actions in which both participants variously assume the roles of addressee and presenter. To carry out joint actions that involve meaning and understanding about the task—and hence, constitute language use—the system must be able to carry out its part in subordinate joint actions in cognition with the user that rely on the existence and accumulation of task-related common ground for their conduct. Ultimately, such joint actions can be seen as coordination problems that must be solved on the shared basis of common ground in order to advance the

*The Destination Status window opens to the right of the Mission Composer. Also note that all three destination names are initially shown in the list of destination choices in the Mission Composer.

†"Baselevel" activation values are used.

‡Activation values in ACT-R are subject to decay over time. Consequently it is possible for a destination that has been "looked at" very recently, but only once, to be more active than one that has been looked at a two or three times but several interactions back.

joint activity of the task toward its goals. Within the limited means of its medium for interacting with the user, task model tracing demonstrates that the requirements and computational nature of such coordination problems can indeed be approached through careful task analysis, cognitive modeling, and the framework of a theory of language use as joint action.

4. CONCLUSIONS

The high-level goal of this report has been to raise and examine the notion that human-computer interaction can and should be viewed as if it were a process of language use. When people do things together, they must act not only as individuals, but also as participants. To accomplish even the smallest of their social purposes, they must find a way and agree to coordinate their abilities. Each person's part in this process is essential, and what emerges from their cooperation is not just the sum of their individual efforts, but a joint action in which, for *people*, not only is something done together, but also meaning and understanding are conveyed and construed. Meaning and understanding are the cognitive residue, as it were, of such joint actions, and it is the accumulation of this in the form of common ground, and the cognitive skills that allow people to use it, that makes people's larger joint activities possible. The way people carry out joint actions is the way language is used. In anticipation of such skills, many of the features of joint actions are superficially appropriated in human-computer interaction designs because they are both familiar and easily grasped by users, and because they can demonstrably promote usability. But this same design strategy can also be the source of an interface's usability problems when users are carelessly led into interactions that imply a program's communicative abilities are greater than they are, or more commonly, when one or more of the conditions people ordinarily try to establish when they enter into joint actions—such as a salient basis for what they are supposed to do or what the program can be expected to do—has been overlooked in the design. While a computer program can only simulate the actions of a true participant in a joint activity, it is nevertheless important that designers have a thorough understanding of the full nature of process in joint actions if they are to be justifiably used and reliably modeled in user interfaces.

Even more ambitious then is the impulse to simulate various cognitive abilities in programs and user interfaces with the aim of making them "intelligent." With this development, programs gain the potential to move much closer to meeting the ordinary expectations people bring to their joint activities with each other in face-to-face settings. In joint activities between people, individuals look for evidence of and rely on each other's cognitive skills; in many respects, this is one of the most fundamental tenets of their common ground. Indeed, whenever joint actions between people involve meaning and understanding, participants necessarily carry out subordinate joint actions in cognition. When a speaker or the initiator of a joint action means for something to be understood, he or she must first determine cognitively what that something is and how to signal it. By presenting signals, the speaker then makes an overt proposal to the addressee concerning specific actions they might carry out together. To understand the speaker's meaning and intentions, the addressee must respond, first by identifying the presentation, and then by meeting the cognitive actions behind the presentation with a corresponding set of cognitive actions in which he or she considers the speaker's proposal by recognizing what has been signaled. In prosecuting their parts in these joint actions in cognition, the participants tacitly rely on a number of elements—shared bases—in their common ground, among which are their mutual knowledge of the signals and signs involved, their mutual assumption of each other's cognitive processing skills and knowledge of behavioral conventions, and their mutual awareness of the events leading up to and including the state of their joint activity. And since their joint activity advances through the use of their accumulating common ground, people make a point of using a related set of cognitive skills to try to keep track of what each other is currently aware of. This then is a short list of the cognitive abilities a program must be able to simulate if it is to interact with its users on a cognitive level in a way that is similar to how people carry out joint actions with each other in a language use sense. If the thesis of this document is that a theory of language use as a form of joint action should serve as a set of first principles for the design of human-computer interaction, then its principal corollary—and the chief focus in the task model tracing work previously described—is that in simulating cognition for interactive purposes, it is necessary to computationally model the interaction-related common ground between the user and the system.

4.1 A Review of the Human-Computer Interaction Design Goals for the Task Model Tracing System

In the task model tracing system, the goal has been to explore the possibilities and ramifications of human-computer interaction on a cognitive level through two theoretical frameworks: Clark's (1996b) characterization of

language use as joint action (the basis for the preceding discussion) and ACT-R, Anderson's (1998) computational theory of human cognition. Respectively, these two frameworks, in concert with an ad hoc process of task analysis, have been used to identify the kinds of processes and representations the system should possess and to implement these components computationally.

On the basis of Clark's work, it is claimed here that human-computer interaction can be usefully likened to interactions between people that are characteristic of their use of language (i.e., interactions that rely on meaning and understanding) in the following ways: computer tasks can be modeled as joint activities between the user and the computer, and individual interactions can be modeled as individual joint actions between the same. Ordinarily in the design of software user interfaces, though, it is the nonreflexive, self-directed cognitive components of joint actions that people are so naturally adept at carrying out that are absent in the system's assessment of user inputs and in its computation of responses and addresses. In people, such cognitive acts are carried out in support of their personal representations of their circumstances and, particularly in their joint activities with others, in support of their common ground—the knowledge, beliefs, and suppositions people presume to share with each other about a cooperative activity. Indeed, the raison d'étre for common ground is the ability to carry out the signaling and recognition actions that are necessary for communication. Since this representation must come from somewhere, for people it is a perpetually accumulative process. In conventional user interfaces, though, while an event-loop and all of its support routines manifest how a system uses its representation of its task to respond to user inputs, that representation, some of which is certainly intended to be on common ground with the user, is functionally inert unless a facility for its growth and/or dynamic configuration is incorporated into the design. Given this language use analysis of process and representational shortcomings in human-computer interaction, it follows that a reasonable approach to more cognizant interaction designs (i.e., designs that are better able to leverage people's communicative skills) can be pursued by modeling task-related common ground as an accumulative process carried out by simulations of the cognitive skills needed to maintain and use it. Hence, this is the approach taken in the task model tracing system. In particular, the system's cognitive simulation takes the form of a task model in which the system maintains its own representation of task-related matters and, as a conceptual subset, a shared representation of the same that constitutes its common ground with the user.

Representationally, common ground in the task model is modeled along two dimensions: first, in terms of Clark's formal definition of common ground, which bears on how it is represented (an implementation issue), and second, in terms of Clark's characterization of the three parts of common ground at any moment, which bears on what is represented and how it is used (an identification issue). Specifically, common ground is defined in terms of propositions that are taken by a community to follow from shared bases that are evident to all. And at any moment, this body of common ground in a joint activity can be thought of as being made up of the joint activity's initial common ground, its current state, and the public events related to the activity that have taken place so far.

It is not generally possible for one participant to know or accurately infer what another knows until it has somehow been made public between the two. As an accumulative process then, much of common ground must be established interactively. In practical terms, this means that the system must be able to simulate cognitive processes that alternatively support and use its common ground with the user. These processes, which represent the system's cognitive skills and abilities, can be identified in terms of the roles the system assumes as a participant in task-related joint actions with the user, both addressee and presenter. In the task model tracing system, these processes are referred to respectively as the system's addressee function and its presentation functions, and each is limited to processing task-related matters. The addressee function updates the system's representation of common ground by interpreting and keeping track of the task's joint actions and the task's status. The system's three presentation functions are referred to separately as its "reporting," "advising," and "doing" functions. Each draws on the system's representation of the task, and hence its common ground, in order to present information to the user that is in turn intended to support his or her common ground with the system. Respectively, these functions summarize the status of the task, make task-related recommendations, and carry out recommended actions for the user.

The product of the cognitive processing carried out by the system's addressee and presentation functions can best be characterized in terms of the three parts of common ground. The system's initial common ground with the user can be thought of as being represented in the design and implementation of the system's components, such as its point-and-click interface, its gamelike task, and the format of its presentations. However, much of this can also be

characterized more pragmatically as potential common ground. Consequently, the role of the addressee function as it interprets user presentations is to establish initial and new common ground. The latter takes the form of a record of the task—its public events so far. In addition, by keeping track of the status of the task, the addressee function maintains a representation of the current state of the joint activity. These representations of the three parts of common ground, particularly, the record of events and the status of the task, are then used in mixed fashion by the presentation functions. The summary of the status of the task generated by the reporting function corresponds to a record of the public events so far that, as modeled, are relevant to the current status of the task. Similarly, the recommendations offered by the advisory function are predicated on the status of the task. And, when the doing function carries out a specific recommendation for the user, the system relies on its current representation of common ground to parameterize the action. When the system publicly carries out task related actions in this manner, it also uses its addressee function to interpret and keep track of the consequences.

The decision to largely implement the system's representation of common ground and its simulation of the cognitive processes necessary for its accumulation, maintenance, and use in the form of a cognitive model in ACT-R was based primarily on the premise of an apparent computational correspondence. Indeed, it may be that the cognitive components of joint actions can best be characterized in cognitive modeling terms, but much more study is warranted. Computationally, ACT-R's distinction between procedural and declarative forms of memory is well suited for representing common ground in formal terms. Both shared bases and the propositions they indicate are naturally modeled as chunks in the system's declarative memory. And production rules in the system's procedural memory naturally model inferences. Other characteristics of the theory also play a role in the implementation—conflict sets of production rules that match the current status of the task, for instance, are used to generate the system's advisory lists, and subsymbolic learning is successfully used to represent a limited form of salience. Additionally, the runtime methodology of task model tracing, itself, in which public interaction events are "traced" in the model's representation of the task, is derived from intelligent tutoring research in which ACT-R models are used to guide student learning.

In summary, task model tracing demonstrates how a simulation of interactive cognitive skills can be used in a human-computer interaction design to represent task-related common ground between the user and the system in its user interface. Of particular concern in the effort is the interpretation of the application task as a joint activity between the user and the system in which individual interactions are taken to be joint actions that can be characterized as instances of language use in the sense meant by Clark. Specifically, joint actions in which meaning and understanding play a role necessarily involve subordinate joint actions that participants carry out in cognition. The ultimate goal in simulating such cognitive processing, is to better leverage users' inherent, cognitively based communicative strengths.

4.2 Future Work and Acknowledgements

The notion in task model tracing that a cognitive model can be used to simulate cognitive functions in a user interface began naively as a solution in search of a problem. On its face, the idea appears to be quite plausible, but in practice, it has proven to be riddled with difficulties, not the least of which is that, despite its many great strides, cognitive modeling is not an applied discipline but an active and many threaded area of research in cognitive science. ACT-R's characterization of the cognitive architecture is just one of a number of competing theories, each with a different set of strengths and weaknesses. Among the practical difficulties faced in task model tracing are problems of scalability—modeling even a portion of the mission planning task has proved to be a highly difficult and iterative process to get right—and representation. It was the latter problem that led to Clark's work when it became apparent that a principled characterization of interaction at the level of cognition was needed as a framework to guide the identification of the task model's declarative and procedural elements in the task analysis process. Far deeper problems, such as procedural learning by example or analogy and the arbitrary representation of a task's state—current areas of weakness in ACT-R—have been left as concerns to be addressed at a later time. Certainly, one of the most conspicuous deficiencies in the system as presented is its lack of a workable solution to the problem of undoing an action.

Indeed, much remains to be done. Recent architectural refinements in ACT-R point to the need to revisit the organization and design of the task model with the aim of reconciling the model with new developments in the theory. In addition, it will be valuable to develop interactive task models of common ground in other tasks for what can be learned about regularities in the process. Ongoing work on collaboration in human-computer interaction in the

field of computational linguistics (e.g., Traum, 1998) is actively concerned with models of common ground, and much of this material remains to be investigated for what it can contribute to future task model designs. Certainly, a more comprehensive theory of the computational representation of salience is needed. Empirical studies of the effectiveness of the presentation functions used in the task model tracing system are clearly warranted, and other interaction and dialogue models for establishing common ground should be investigated. Last, knowledge elicitation remains a problem for application programming interfaces for purposes of interaction with cognitive simulations, in part because of a lack of representation standards, and more work certainly could to be done in this area.

As a final note, despite all difficulties, it is my conviction that much of the theory underpinning the task model tracing work presented in this report is fundamentally sound. Contemporary software user interfaces place the user in a face-to-face setting in which task-related interactions can easily and convincingly be characterized as joint projects, especially when the increasingly substantial computational power behind them is taken into account. If the notion of a system's usability is a measure of how well its design accommodates the user's human strengths, then surely future interaction designs will be keenly concerned with system-level models of interaction in cognition.

This work was supported by the Office of Naval Research. I would also like to offer my deepest thanks to James Ballas, Helen Gigley, John Sibert, Wayne Gray, Dianne Martin, and Greg Trafton for their input and encouragement.

REFERENCES

Anderson, J. R., and Lebiere, C. (1998). *The Atomic Components of Thought*. Mahwah, NJ: Lawrence Erlbaum Associates.

Anderson, J. R. (1987). Production system, learning, and tutoring. In *Production System Models of Learning and Development*. Cambridge, MA: MIT Press.

Anderson, J. R. (1990). *The Adaptive Character of Thought*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, J. R. (1993). *Rules of the Mind*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Austin, J. L. (1962). *How to Do Things with Words*. Cambridge, MA: Harvard University Press.

Clark, H. H. (1996a). Opening plenary: Arranging to do things with others. In *Proceedings of the CHI '96 Conference Companion on Human Factors in Computing Systems: Common Ground*. Association for Computing Machinery. 165–167.

Clark, H. H. (1996b). *Using Language*. New York, NY: Cambridge University Press.

Cohen, R., Allaby, C., Cumbaa, C., Fitzgerald, M., Ho, K., Hui, B., Latulipe, C., Lu, F., Moussa, N., Pooley, D., Qian, A., and Siddiqi, S. (1998). What is initiative? *User Modeling and User-Adapted Interaction* 8:171–214.

Erickson, T. D. (1990). Working with interface metaphors. In Laurel, B., ed., *The Art of Human-Computer Interface Design*. Reading, MA: Addison Wesley.

Grice, H. P. (1957). Meaning. *Philosophical Review* 66:377–388.

Horvitz, E., Breese, J., Heckerman, D., Hovel, D., and Rommelse, K. (1998). The Lumiere Project: Baysian user modeling for inferring the goals and needs of software users. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. San Francisco, CA: Morgan-Kaufman.

Kay, A. (1984). Computer software. *Scientific American*. (September), p.52.

Klahr, D., Langley, P., and Neches, R., eds. (1987). *Production System Models of Learning and Development*. Cambridge, MA: MIT Press.

Laurel, B., ed. (1990). *The Art of Human-Computer Interface Design*. Reading, MA: Addison Wesley.

Maybury, M. T., ed. (1993). *Intelligent MultiMedia Interfaces*. Cambridge, MA: AAAI Press.

McFarlane, D. C. (1998). *Interruption of People in Human-Computer Interaction*. Ph.D. Dissertation, The George Washington University.

Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.

Norman, D. A. (1986). Cognitive engineering. In Norman, D. A., and Draper, S. W., eds., *User Centered System Design*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Norman, D. A. (1988). *The Design of Everyday Things*. New York, NY: Doubleday.

Norman, D. A. (1992). *Turn Signals Are the Facial Expressions of Automobiles*. Reading, MA: Addison-Wesley.

Rich, C., and Sidner, C. L. (1998). COLLAGEN: a collaboration manager for software interface agents. *User Modeling and User-Adapted Interaction* 8:315–350.

Ritter, F. E., and Major, N. P. (1995). Useful mechanisms for developing simulations for cognitive models. *AISB Quarterly* 91:7–18.

Sellen, A., and Nicol, A. (1990). Building user-centered on-line help. In Laurel, B., ed., *The Art of Human-Computer Interface Design*. Reading, MA: Addison Wesley.

Shneiderman, B. (1998). *Designing the User Interface: Stategies for Effective Human-Computer Interaction*. Reading, MA: Addison-Wesley, 3rd edition.

Simon, H. A., and Kaplan, C. A. (1989). Foundations of cognitive science. In Posner, M. I., ed., *Foundations of Cognitive Science*. Cambridge, MA: MIT Press.

Sullivan, J. W., and Tyler, S. W., eds. (1991). *Intelligen User Interfaces*. New York, NY: ACM Press.

Tognazzini, B. (1990). Consistency. In Laurel, B., ed., *The Art of Human-Computer Interface Design*. Reading, MA: Addison Wesley.

Traum, D. R. (1998). On Clark and Schaefer's contribution model and its applicability to human-computer collaboration. In *Proceedings of the Third International Conference on the Design of Cooperative Systems (COOP'98)*. Rocquencourt, France: INRIA Press. (Clark's work was sole focus of the workshop at which this paper was given.).